



# Red Hat Enterprise Linux 7

## SystemTap タップセットリファレンス

Red Hat Enterprise Linux 7 SystemTap 用



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

---

Red Hat Enterprise Linux 7 SystemTap 用

Robert Krátký  
Red Hat Customer Content Services  
rkratky@redhat.com

William Cohen  
Red Hat Software Engineering

Don Domingo  
Red Hat Customer Content Services

Jacquelynn East  
Red Hat Customer Content Services

Red Hat Enterprise Linux Documentation

## 法律上の通知

Copyright © 2010 Red Hat, Inc. and others.

This document is licensed by Red Hat under the [Creative Commons Attribution-ShareAlike 3.0 Unported License](#). If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## 概要

タップセットリファレンスガイドは、ユーザーが **SystemTap** スクリプトに適用できる最も一般的なタップセット定義について説明しています。

---

目次

第1章 はじめに .....	52
1.1. 本ガイドの目的 .....	52
第2章 タップセット開発ガイドライン .....	53
2.1. タップセットの記述 .....	53
2.2. タップセットの要素 .....	54
第3章 コンテキスト関数 .....	57
名前 .....	57
概要 .....	57
引数 .....	57
説明 .....	57
名前 .....	57
概要 .....	57
引数 .....	57
説明 .....	57
名前 .....	57
概要 .....	57
引数 .....	57
説明 .....	57
名前 .....	58
概要 .....	58
引数 .....	58
説明 .....	58
名前 .....	58
概要 .....	58
引数 .....	58
説明 .....	58
名前 .....	58
概要 .....	58
引数 .....	58
説明 .....	59
名前 .....	59
概要 .....	59
引数 .....	59
説明 .....	59
名前 .....	59
概要 .....	59
引数 .....	59
説明 .....	59
名前 .....	60
概要 .....	60
引数 .....	60
説明 .....	60
名前 .....	60
概要 .....	60
引数 .....	60
説明 .....	60
名前 .....	60
概要 .....	60
引数 .....	60
説明 .....	60

[illegible]

概要	65
引数	65
説明	65
名前	65
概要	65
引数	65
説明	65
名前	65
概要	65
引数	66
説明	66
名前	66
概要	66
引数	66
説明	66
名前	66
概要	66
引数	66
説明	66
名前	66
概要	66
引数	67
説明	67
名前	67
概要	67
引数	67
説明	67
名前	67
概要	67
引数	67
説明	67
名前	67
概要	67
引数	67
説明	68
名前	68
概要	68
引数	68
説明	68
名前	68
概要	68
引数	68
説明	68
名前	68
概要	68
引数	68
説明	68
名前	69
概要	69
引数	69
説明	69
名前	69
概要	69

引数	69
説明	69
名前	69
概要	69
引数	69
説明	70
名前	70
概要	70
引数	70
説明	70
名前	70
概要	70
引数	70
説明	70
コンテキスト	70
名前	70
概要	70
引数	70
説明	71
名前	71
概要	71
引数	71
説明	71
名前	71
概要	71
引数	71
説明	71
名前	71
概要	71
引数	71
説明	72
名前	72
概要	72
引数	72
説明	72
名前	72
概要	72
引数	72
説明	72
名前	72
概要	72
引数	72
説明	73
注記	73
名前	73
概要	73
引数	73
説明	73
名前	73
概要	73
引数	73
説明	73
注記	73



名前	74
概要	74
引数	74
説明	74
注記	74
名前	74
概要	74
引数	74
説明	74
注記	74
名前	75
概要	75
引数	75
説明	75
名前	75
概要	75
引数	75
説明	75
名前	75
概要	75
引数	75
説明	75
名前	75
概要	75
引数	75
説明	76
注記	76
名前	76
概要	76
引数	76
説明	76
名前	76
概要	76
引数	76
説明	76
名前	76
概要	76
引数	76
説明	76
名前	76
概要	77
引数	77
説明	77
名前	77
概要	77
引数	77
説明	77
名前	77
概要	77
引数	77
説明	77
名前	77
概要	78
引数	78
説明	78
名前	78
概要	78
引数	78
説明	78
名前	78
概要	78
引数	78
説明	78
名前	79
概要	79

引数	79
説明	79
名前	79
概要	79
引数	79
説明	79
名前	79
概要	79
引数	79
説明	80
名前	80
概要	80
引数	80
説明	80
名前	80
概要	80
引数	80
説明	80
注記	81
名前	81
概要	81
引数	81
説明	81
名前	81
概要	81
引数	81
説明	81
注記	82
名前	82
概要	82
引数	82
説明	82
注記	82
名前	82
概要	82
引数	82
説明	82
名前	83
概要	83
引数	83
説明	83
名前	83
概要	83
引数	83
説明	83
名前	83
概要	84
引数	84
説明	84
名前	84
概要	84
引数	84
説明	84

名前	84
概要	84
引数	84
説明	84
名前	84
概要	84
引数	85
説明	85
名前	85
概要	85
引数	85
説明	85
名前	85
概要	85
引数	85
説明	85
名前	86
概要	86
引数	86
説明	86
名前	86
概要	86
引数	86
説明	86
名前	86
概要	86
引数	86
説明	86
名前	87
概要	87
引数	87
説明	87
名前	87
概要	87
引数	87
説明	87
名前	87
概要	88
引数	88
説明	88
名前	88
概要	88
引数	88
説明	88
名前	88
概要	88
引数	88
名前	88
概要	89
引数	89
説明	89
名前	89
概要	89

引数	89
説明	89
名前	89
概要	89
引数	89
名前	89
概要	90
引数	90
説明	90
名前	90
概要	90
引数	90
説明	90
名前	90
概要	90
引数	90
説明	90
名前	91
概要	91
引数	91
説明	91
名前	91
概要	91
引数	91
説明	91
名前	91
概要	91
引数	91
説明	91
名前	91
概要	91
引数	91
説明	92
名前	92
概要	92
引数	92
説明	92
名前	92
概要	92
引数	92
説明	92
名前	92
概要	92
引数	92
説明	92
名前	92
概要	92
引数	93
説明	93
名前	93
概要	93
引数	93
説明	93
名前	93
概要	93
引数	93
説明	93
名前	93
概要	93
引数	93
説明	93
名前	93
概要	94
引数	94
説明	94

名前	94
概要	94
引数	94
説明	94
名前	94
概要	94
引数	94
説明	94
名前	95
概要	95
引数	95
説明	95
名前	95
概要	95
引数	95
説明	95
名前	95
概要	95
引数	95
説明	95
名前	95
概要	95
引数	95
説明	96
名前	96
概要	96
引数	96
説明	96
名前	96
概要	96
引数	96
説明	96
名前	96
概要	96
引数	96
説明	96
名前	96
概要	96
引数	96
説明	97
名前	97
概要	97
引数	97
説明	97
名前	97
概要	97
引数	97
説明	97
名前	97
概要	97
引数	97
説明	97
名前	97
概要	97
引数	97
説明	98
名前	98
概要	98
引数	98
説明	98
注記	98
名前	98
概要	98
引数	98
説明	98

注記	98
名前	99
概要	99
引数	99
説明	99
名前	99
概要	99
引数	99
説明	99
名前	99
概要	99
引数	99
説明	100
名前	100
概要	100
引数	100
説明	100
名前	100
概要	100
引数	100
説明	100
名前	100
概要	100
引数	100
説明	100
名前	100
概要	100
引数	101
説明	101
名前	101
概要	101
引数	101
説明	101
名前	101
概要	101
引数	101
説明	101
名前	101
概要	101
引数	101
説明	101
名前	101
概要	102
引数	102
説明	102
名前	102
概要	102
引数	102
説明	102
名前	102
概要	102
引数	102
説明	102
名前	102
概要	102
引数	102
説明	102
名前	103
概要	103
引数	103
説明	103
第4章 タイムスタンプ関数 .....	104
名前	104
概要	104

引数	104
説明	104
名前	104
概要	104
引数	104
説明	104
名前	104
概要	104
引数	104
説明	105
名前	105
概要	105
引数	105
説明	105
名前	105
概要	105
引数	105
説明	105
名前	105
概要	105
引数	105
説明	105
名前	105
概要	106
引数	106
説明	106
名前	106
概要	106
引数	106
説明	106
名前	106
概要	106
引数	106
説明	106
名前	106
概要	106
引数	106
説明	106
名前	106
概要	107
引数	107
説明	107
名前	107
概要	107
引数	107
説明	107
名前	107
概要	107
引数	107
説明	107
名前	107
概要	107
引数	107
説明	107
名前	107
概要	107
引数	108
説明	108
名前	108
概要	108
引数	108
説明	108
名前	108
概要	108
引数	108

説明	108
名前	108
概要	108
引数	108
説明	109
名前	109
概要	109
引数	109
説明	109
名前	109
概要	109
引数	109
説明	109
名前	109
概要	109
引数	109
説明	110
名前	110
概要	110
引数	110
説明	110
名前	110
概要	110
引数	110
説明	110
名前	110
概要	111
引数	111
説明	111
名前	111
概要	111
引数	111
説明	111
<b>第5章 時間ユーティリティ関数 .....</b>	<b>112</b>
名前	112
概要	112
引数	112
説明	112
名前	112
概要	112
引数	113
説明	113
名前	113
概要	113
引数	113
説明	113
名前	113
概要	113
引数	113
説明	113
<b>第6章 シェルコマンド関数 .....</b>	<b>114</b>



名前	114
概要	114
引数	114
説明	114
<b>第7章 メモリータップセット .....</b>	<b>115</b>
名前	115
概要	115
引数	115
説明	115
名前	115
概要	115
引数	115
説明	115
名前	115
概要	115
引数	116
名前	116
概要	116
引数	116
説明	116
名前	116
概要	116
引数	116
説明	116
名前	116
概要	116
引数	116
説明	117
名前	117
概要	117
引数	117
説明	117
名前	117
概要	117
引数	117
説明	117
名前	117
概要	117
引数	118
説明	118
名前	118
概要	118
引数	118
説明	118
名前	118
概要	118
引数	118
説明	118
名前	118
概要	118
引数	119
説明	119

名前	119
概要	119
引数	119
説明	119
名前	119
概要	119
引数	119
説明	119
名前	119
概要	120
引数	120
説明	120
名前	120
概要	120
引数	120
説明	120
名前	120
概要	120
引数	120
説明	120
名前	121
概要	121
値	121
コンテキスト	121
名前	121
概要	121
値	121
名前	122
概要	122
値	122
名前	122
概要	122
値	122
名前	123
概要	123
値	123
名前	124
概要	124
値	124
名前	125
概要	125
値	125
名前	125
概要	125
値	125
コンテキスト	125
名前	126
概要	126
値	126
コンテキスト	126
名前	126
概要	126
値	126
コンテキスト	126

名前	126
概要	126
値	127
コンテキスト	127
名前	127
概要	127
値	127
名前	127
概要	127
値	127
コンテキスト	128
説明	128
名前	128
概要	128
値	128
コンテキスト	128
説明	128
<b>第8章 タスク時間タプセット .....</b>	<b>129</b>
名前	129
概要	129
引数	129
名前	129
概要	129
引数	129
説明	129
名前	129
概要	129
引数	129
名前	130
概要	130
引数	130
説明	130
名前	130
概要	130
引数	130
説明	130
名前	130
概要	130
引数	130
説明	131
名前	131
概要	131
引数	131
説明	131
名前	131
概要	131
引数	131
説明	131
名前	131
概要	131
引数	131
説明	132
名前	132
概要	132
引数	132
説明	132

名前	132
概要	132
引数	132
説明	132
名前	132
概要	132
引数	132
説明	132
名前	132
概要	132
引数	133
説明	133
名前	133
概要	133
引数	133
説明	133
<b>第9章 スケジューラータップセット .....</b>	<b>134</b>
名前	134
概要	134
値	134
コンテキスト	134
名前	134
概要	134
値	134
コンテキスト	134
名前	135
概要	135
値	135
コンテキスト	135
名前	135
概要	135
値	135
名前	136
概要	136
値	136
名前	136
概要	136
値	136
名前	137
概要	137
値	137
名前	137
概要	137
値	137
名前	138
概要	138
値	138
名前	138
概要	138
値	138
名前	139
概要	139

値	139
名前	139
概要	139
値	139
名前	139
概要	139
値	140
コンテキスト	140
名前	140
概要	140
値	140
名前	140
概要	140
値	140
名前	141
概要	141
値	141
<b>第10章 IO スケジューラーおよびブロック IO タップセット .....</b>	<b>142</b>
名前	142
概要	142
値	142
コンテキスト	143
名前	143
概要	143
値	143
コンテキスト	144
名前	144
概要	144
値	144
コンテキスト	145
名前	145
概要	145
値	146
コンテキスト	147
名前	147
概要	147
値	147
コンテキスト	148
名前	148
概要	148
値	148
名前	149
概要	149
値	149
名前	149
概要	149
値	150
名前	150
概要	150
値	150
名前	151
概要	151

値	151
名前	151
概要	151
値	151
名前	152
概要	152
値	152
名前	152
概要	152
値	152
説明	153
名前	153
概要	153
値	153
説明	154
名前	154
概要	154
値	154
説明	154
名前	154
概要	154
値	154
説明	155
名前	155
概要	155
値	155
説明	155
名前	155
概要	155
値	155
説明	155
<b>第11章 SCSI タップセット .....</b>	<b>157</b>
名前	157
概要	157
値	157
名前	158
概要	158
値	158
名前	159
概要	159
値	159
名前	159
概要	160
値	160
名前	160
概要	160
値	160
名前	161
概要	161
値	161
<b>第12章 TTY タップセット .....</b>	<b>163</b>

名前	163
概要	163
値	163
名前	163
概要	163
値	163
名前	163
概要	164
値	164
名前	164
概要	164
値	164
名前	164
概要	165
値	165
名前	165
概要	165
値	165
名前	166
概要	166
値	166
名前	166
概要	166
値	166
名前	167
概要	167
値	167
名前	168
概要	168
値	168
名前	168
概要	168
値	168
<b>第13章 割り込み要求 (IRQ) タップセット .....</b>	<b>170</b>
名前	170
概要	170
値	170
名前	171
概要	171
値	171
名前	172
概要	172
値	172
名前	172
概要	172
値	172
名前	173
概要	173
値	173
名前	173
概要	173
値	173

名前	173
概要	173
値	174
名前	174
概要	174
値	174
第14章 ネットワーキングタプセット .....	175
名前	175
概要	175
引数	175
名前	175
概要	175
引数	175
名前	175
概要	175
引数	175
名前	176
概要	176
引数	176
名前	176
概要	176
引数	176
名前	176
概要	176
引数	176
名前	176
概要	177
引数	177
名前	177
概要	177
引数	177
名前	177
概要	177
値	177
名前	178
概要	178
値	178
名前	178
概要	178
値	178
名前	178
概要	178
値	178
名前	179
概要	179
値	179
名前	179
概要	179
値	179
名前	179
概要	179
値	180



名前	180
概要	180
値	180
名前	180
概要	180
値	180
名前	181
概要	181
値	181
名前	181
概要	181
値	181
名前	181
概要	181
値	181
名前	182
概要	182
値	182
名前	182
概要	182
値	182
名前	182
概要	182
値	182
名前	182
概要	183
値	183
名前	184
概要	184
値	184
名前	186
概要	186
値	186
名前	188
概要	188
値	188
名前	190
概要	190
値	190
名前	192
概要	192
値	192
名前	194
概要	194
値	194
名前	196
概要	197
値	197
名前	199
概要	199
値	199
名前	201
概要	201
値	201
名前	203
概要	203

値	203
名前	205
概要	205
値	205
名前	207
概要	207
値	207
名前	209
概要	209
値	209
名前	209
概要	209
値	210
名前	210
概要	210
値	210
名前	211
概要	211
値	211
名前	212
概要	212
値	212
名前	213
概要	213
値	213
名前	213
概要	213
値	213
名前	215
概要	215
値	215
名前	215
概要	215
値	216
名前	216
概要	216
値	216
名前	217
概要	217
値	217
説明	217
名前	217
概要	217
値	217
名前	218
概要	218
値	218
名前	219
概要	219
値	219
名前	219
概要	219
値	219

名前	220
概要	220
値	220
名前	221
概要	221
値	221
説明	221
名前	221
概要	221
値	221
名前	222
概要	222
値	222
コンテキスト	223
名前	223
概要	223
値	223
コンテキスト	223
名前	223
概要	223
値	223
名前	224
概要	224
値	224
コンテキスト	225
名前	225
概要	225
値	225
コンテキスト	226
名前	226
概要	226
値	226
コンテキスト	226
名前	226
概要	226
値	227
コンテキスト	227
名前	227
概要	227
値	227
コンテキスト	227
名前	228
概要	228
値	228
コンテキスト	228
名前	228
概要	228
値	228
コンテキスト	229
名前	229
概要	229
値	229
コンテキスト	229

名前	230
概要	230
値	230
コンテキスト	230
名前	230
概要	230
値	230
コンテキスト	231
名前	231
概要	231
値	231
コンテキスト	232
名前	232
概要	232
値	232
コンテキスト	232
<b>第15章 ソケットタップセット .....</b>	<b>233</b>
名前	233
概要	233
引数	233
名前	233
概要	233
引数	233
名前	233
概要	233
引数	233
名前	234
概要	234
引数	234
名前	234
概要	234
引数	234
名前	234
概要	234
引数	234
名前	234
概要	235
引数	235
名前	235
概要	235
引数	235
名前	235
概要	235
値	235
コンテキスト	236
説明	236
名前	236
概要	236
値	236
コンテキスト	237
説明	237
名前	237

概要	237
値	237
コンテキスト	237
説明	237
名前	238
概要	238
値	238
コンテキスト	238
説明	238
名前	238
概要	238
値	239
コンテキスト	239
説明	239
名前	239
概要	239
値	239
コンテキスト	239
説明	239
名前	240
概要	240
値	240
コンテキスト	240
説明	240
名前	240
概要	240
値	240
コンテキスト	241
説明	241
名前	241
概要	241
値	241
コンテキスト	242
説明	242
名前	242
概要	242
値	242
コンテキスト	243
説明	243
名前	243
概要	243
値	243
コンテキスト	243
説明	243
名前	244
概要	244
値	244
コンテキスト	244
説明	244
名前	244
概要	244
値	245
コンテキスト	245

名前	245
概要	245
値	245
コンテキスト	246
説明	246
名前	246
概要	246
値	246
コンテキスト	247
説明	247
名前	247
概要	247
値	247
コンテキスト	248
名前	248
概要	248
値	248
コンテキスト	248
説明	248
名前	249
概要	249
値	249
コンテキスト	249
説明	249
名前	249
概要	249
値	250
コンテキスト	250
説明	250
名前	250
概要	250
値	250
コンテキスト	251
説明	251
名前	251
概要	251
値	251
コンテキスト	252
説明	252
名前	252
概要	252
値	252
コンテキスト	253
説明	253
<b>第16章 SNMP 情報タップセット .....</b>	<b>254</b>
名前	254
概要	254
引数	254
説明	254
名前	254
概要	254
引数	254

説明	254
名前	255
概要	255
引数	255
説明	255
名前	255
概要	255
引数	255
説明	255
名前	255
概要	255
引数	256
説明	256
名前	256
概要	256
引数	256
説明	256
名前	256
概要	256
引数	256
説明	257
名前	257
概要	257
引数	257
説明	257
名前	257
概要	257
引数	257
説明	257
名前	257
概要	258
引数	258
説明	258
名前	258
概要	258
引数	258
説明	258
名前	258
概要	258
引数	258
説明	258
名前	259
概要	259
引数	259
説明	259
名前	259
概要	259
値	259
説明	259
名前	259
概要	259
値	259
説明	260

名前	260
概要	260
値	260
説明	260
名前	260
概要	260
値	260
説明	261
名前	261
概要	261
値	261
説明	261
名前	261
概要	261
値	261
説明	261
名前	262
概要	262
値	262
説明	262
名前	262
概要	262
値	262
説明	262
名前	263
概要	263
値	263
説明	263
名前	263
概要	263
値	263
説明	263
名前	263
概要	263
値	264
説明	264
名前	264
概要	264
値	264
説明	264
名前	264
概要	264
値	264
説明	265
名前	265
概要	265
値	265
説明	265
名前	265
概要	265
値	265
説明	266
名前	266



概要	266
値	266
説明	266
名前	266
概要	266
値	266
説明	266
名前	267
概要	267
値	267
説明	267
名前	267
概要	267
値	267
説明	267
名前	268
概要	268
値	268
説明	268
名前	268
概要	268
値	268
説明	268
名前	268
概要	269
値	269
説明	269
名前	269
概要	269
値	269
説明	269
名前	269
概要	269
値	270
説明	270
<b>第17章 カーネルプロセスタップセット .....</b>	<b>271</b>
名前	271
概要	271
引数	271
説明	271
名前	271
概要	271
引数	271
説明	271
名前	271
概要	271
引数	271
説明	272
名前	272
概要	272
引数	272
説明	272

名前	272
概要	272
値	272
コンテキスト	272
説明	272
名前	273
概要	273
値	273
コンテキスト	273
説明	273
名前	273
概要	273
値	273
コンテキスト	274
説明	274
名前	274
概要	274
値	274
コンテキスト	274
説明	274
名前	274
概要	274
値	274
コンテキスト	275
説明	275
名前	275
概要	275
値	275
コンテキスト	275
説明	275
<b>第18章 シグナルタップセット .....</b>	<b>276</b>
名前	276
概要	276
引数	276
名前	276
概要	276
引数	276
名前	276
概要	276
引数	276
名前	277
概要	277
引数	277
名前	277
概要	277
引数	277
説明	277
名前	277
概要	277
引数	277
名前	278
概要	278

引数	278
名前	278
概要	278
値	278
名前	278
概要	278
値	278
名前	279
概要	279
値	279
名前	279
概要	280
値	280
名前	280
概要	280
値	280
名前	281
概要	281
値	281
名前	281
概要	281
値	281
名前	281
概要	281
値	281
名前	281
概要	281
値	282
名前	282
概要	282
値	282
名前	282
概要	282
値	282
名前	283
概要	283
値	283
説明	283
名前	284
概要	284
値	284
説明	284
名前	284
概要	284
値	284
名前	284
概要	284
値	285
名前	285
概要	285
値	285
名前	285
概要	285
値	286
コンテキスト	286
名前	286

概要	286
値	286
コンテキスト	287
説明	287
名前	287
概要	287
値	287
名前	288
概要	288
値	288
名前	288
概要	288
値	288
説明	289
名前	289
概要	289
値	289
名前	289
概要	289
値	289
説明	290
名前	290
概要	290
値	290
名前	291
概要	291
値	291
名前	291
概要	291
値	291
名前	291
概要	291
値	291
第19章 ERRNO タップセット .....	293
名前	293
概要	293
引数	293
説明	293
名前	293
概要	293
引数	293
説明	293
名前	293
概要	294
引数	294
説明	294
名前	294
概要	294
引数	294
説明	294
第20章 RLIMIT タップセット .....	295

名前	295
概要	295
引数	295
説明	295
<b>第21章 デバイスタップセット .....</b>	<b>296</b>
名前	296
概要	296
引数	296
名前	296
概要	296
引数	296
名前	296
概要	296
引数	296
名前	297
概要	297
引数	297
<b>第22章 DIRECTORY-ENTRY (DENTRY) タップセット .....</b>	<b>298</b>
名前	298
概要	298
引数	298
説明	298
名前	298
概要	298
引数	298
説明	298
名前	298
概要	298
引数	298
説明	299
名前	299
概要	299
引数	299
説明	299
名前	299
概要	299
引数	299
説明	299
名前	299
概要	300
引数	300
説明	300
名前	300
概要	300
引数	300
説明	300
名前	300
概要	300
引数	300
説明	300
名前	300
概要	300
引数	300
説明	300
名前	301

概要	301
引数	301
説明	301
名前	301
概要	301
引数	301
説明	301
<b>第23章 ログイングタップセット .....</b>	<b>302</b>
名前	302
概要	302
引数	302
説明	302
名前	302
概要	302
引数	302
説明	302
名前	302
概要	303
引数	303
説明	303
名前	303
概要	303
引数	303
説明	303
名前	303
概要	303
引数	303
説明	303
名前	304
概要	304
引数	304
説明	304
名前	304
概要	304
引数	304
説明	304
<b>第24章 キューに関連する統計タップセット .....</b>	<b>305</b>
名前	305
概要	305
引数	305
説明	305
名前	305
概要	305
引数	305
説明	305
名前	305
概要	305
引数	305
説明	306
名前	306
概要	306

引数	306
説明	306
名前	306
概要	306
引数	306
説明	306
該当するキューの統計	306
名前	307
概要	307
引数	307
説明	307
名前	307
概要	307
引数	307
説明	307
名前	307
概要	307
引数	307
説明	308
名前	308
概要	308
引数	308
説明	308
名前	308
概要	308
引数	308
説明	308
名前	309
概要	309
引数	309
説明	309
<b>第25章 ランダム関数タプセット .....</b>	<b>310</b>
名前	310
概要	310
引数	310
<b>第26章 文字列およびデータ取得関数タプセット .....</b>	<b>311</b>
名前	311
概要	311
引数	311
説明	311
名前	311
概要	311
引数	311
説明	311
名前	311
概要	311
引数	312
説明	312
名前	312
概要	312
引数	312

説明	312
名前	312
概要	312
引数	312
説明	312
名前	313
概要	313
引数	313
説明	313
名前	313
概要	313
引数	313
説明	313
名前	313
概要	313
引数	313
説明	314
名前	314
概要	314
引数	314
説明	314
名前	314
概要	314
引数	314
説明	314
名前	315
概要	315
引数	315
説明	315
名前	315
概要	315
引数	315
説明	315
名前	315
概要	315
引数	316
説明	316
名前	316
概要	316
引数	316
説明	316
名前	316
概要	316
引数	316
説明	316
名前	317
概要	317
引数	317
説明	317
名前	317
概要	317
引数	317
説明	317



名前	317
概要	317
引数	318
説明	318
名前	318
概要	318
引数	318
説明	318
名前	318
概要	318
引数	318
説明	318
名前	319
概要	319
引数	319
説明	319
名前	319
概要	319
引数	319
説明	319
名前	319
概要	319
引数	319
説明	319
名前	319
概要	319
引数	319
説明	320
名前	320
概要	320
引数	320
説明	320
名前	320
概要	320
引数	320
説明	320
名前	320
概要	320
引数	321
説明	321
名前	321
概要	321
引数	321
説明	321
名前	321
概要	321
引数	321
説明	321
名前	322
概要	322
引数	322
説明	322
名前	322
概要	322
引数	322
説明	322
名前	322

概要	322
引数	322
説明	323
名前	323
概要	323
引数	323
説明	323
名前	323
概要	323
引数	323
説明	324
名前	324
概要	324
引数	324
説明	324
名前	324
概要	324
引数	324
説明	324
名前	324
概要	324
引数	324
説明	324
名前	325
概要	325
引数	325
説明	325
名前	325
概要	325
引数	325
説明	325
名前	326
概要	326
引数	326
説明	326
名前	326
概要	326
引数	326
説明	326
名前	327
概要	327
引数	327
説明	327
名前	327
概要	327
引数	327
説明	327
名前	327
概要	327
引数	328
説明	328
名前	328
概要	328
引数	328
説明	328
名前	328
概要	328

引数	328
説明	328
名前	329
概要	329
引数	329
説明	329
名前	329
概要	329
引数	329
説明	329
名前	329
概要	329
引数	329
説明	330
名前	330
概要	330
引数	330
説明	330
名前	330
概要	330
引数	330
説明	330
名前	330
概要	330
引数	331
説明	331
名前	331
概要	331
引数	331
説明	331
名前	331
概要	331
引数	331
説明	331
名前	332
概要	332
引数	332
説明	332
名前	332
概要	332
引数	332
説明	332
<b>第27章 文字列およびデータ書き込み関数タプセット .....</b>	<b>333</b>
名前	333
概要	333
引数	333
説明	333
名前	333
概要	333
引数	333
説明	333
名前	334

概要	334
引数	334
説明	334
名前	334
概要	334
引数	334
説明	334
名前	334
概要	334
引数	335
説明	335
名前	335
概要	335
引数	335
説明	335
名前	335
概要	335
引数	335
説明	336
<b>第28章 GURU タップセット</b> .....	<b>337</b>
名前	337
概要	337
引数	337
説明	337
名前	337
概要	337
引数	337
説明	337
名前	337
概要	337
引数	338
説明	338
名前	338
概要	338
引数	338
説明	338
<b>第29章 標準的な文字列関数のコレクション</b> .....	<b>339</b>
名前	339
概要	339
引数	339
説明	339
名前	339
概要	339
引数	339
説明	339
名前	339
概要	340
引数	340
説明	340
名前	340
概要	340

引数	340
説明	340
名前	340
概要	340
引数	340
説明	341
名前	341
概要	341
引数	341
説明	341
名前	341
概要	341
引数	341
説明	341
名前	342
概要	342
引数	342
説明	342
名前	342
概要	342
引数	342
説明	342
名前	342
概要	343
引数	343
説明	343
名前	343
概要	343
引数	343
説明	343
<b>第30章 ANSI 制御文字をログで使用するためのユーティリティー関数 .....</b>	<b>344</b>
名前	344
概要	344
引数	344
説明	344
名前	344
概要	344
引数	344
説明	344
名前	344
概要	344
引数	344
説明	345
名前	345
概要	345
引数	345
説明	345
名前	345
概要	345
引数	345
説明	345
名前	345

概要	345
引数	345
説明	346
名前	346
概要	346
引数	346
説明	346
名前	346
概要	346
引数	346
説明	346
名前	346
概要	346
引数	346
説明	346
名前	347
概要	347
引数	347
説明	347
名前	347
概要	347
引数	347
説明	347
名前	348
概要	348
引数	348
説明	348
名前	348
概要	348
引数	348
説明	348
名前	348
概要	348
引数	349
説明	349
名前	349
概要	349
引数	349
説明	349
<b>第31章 SYSTEMTAP トランスレータータップセット .....</b>	<b>350</b>
名前	350
概要	350
値	350
説明	350
名前	350
概要	350
値	350
説明	350
名前	351
概要	351
値	351
説明	351

名前	351
概要	351
値	351
説明	351
名前	351
概要	351
値	352
説明	352
名前	352
概要	352
値	352
説明	352
名前	352
概要	352
値	352
説明	352
名前	353
概要	353
値	353
説明	353
名前	353
概要	353
値	353
説明	353
名前	353
概要	353
値	353
説明	354
名前	354
概要	354
値	354
説明	354
名前	354
概要	354
値	354
説明	354
名前	354
概要	354
値	354
説明	355
名前	355
概要	355
値	355
説明	355
名前	355
概要	355
値	355
説明	355
名前	355
概要	356
値	356
説明	356
名前	356

概要	356
値	356
説明	356
名前	356
概要	356
値	356
説明	356
名前	357
概要	357
値	357
説明	357
名前	357
概要	357
値	357
説明	357
名前	357
概要	357
値	357
説明	357
名前	358
概要	358
値	358
説明	358
名前	358
概要	358
値	358
説明	358
名前	358
概要	358
値	359
説明	359
名前	359
概要	359
値	359
説明	359
名前	359
概要	359
値	359
説明	359
名前	360
概要	360
値	360
説明	360
<b>第32章 ネットワークファイルストレージタップセット .....</b>	<b>361</b>
名前	361
概要	361
引数	361
説明	361
名前	361
概要	361
値	361
説明	362



名前	362
概要	362
値	362
説明	363
名前	363
概要	363
値	363
説明	363
名前	363
概要	363
値	363
説明	364
名前	364
概要	364
値	364
説明	364
名前	365
概要	365
値	365
説明	365
名前	366
概要	366
値	366
説明	367
名前	367
概要	367
値	367
説明	367
名前	368
概要	368
値	368
名前	368
概要	368
値	369
名前	369
概要	369
値	369
名前	369
概要	369
値	370
名前	370
概要	370
値	370
名前	370
概要	370
値	370
名前	371
概要	371
値	371
名前	372
概要	372
値	372
名前	373

概要	373
値	373
名前	373
概要	373
値	373
説明	373
名前	374
概要	374
値	374
名前	374
概要	374
値	375
名前	375
概要	375
値	375
名前	376
概要	376
値	376
説明	376
名前	376
概要	376
値	376
名前	377
概要	377
値	377
説明	377
名前	377
概要	377
値	378
説明	378
名前	378
概要	378
値	378
説明	379
名前	379
概要	379
値	379
名前	380
概要	380
値	380
説明	380
名前	380
概要	380
値	380
名前	381
概要	381
値	381
説明	381
名前	381
概要	381
値	382
説明	382
名前	382

概要	382
値	382
説明	383
名前	383
概要	383
値	383
説明	383
名前	384
概要	384
値	384
説明	384
名前	384
概要	384
値	384
名前	385
概要	385
値	385
名前	386
概要	386
値	386
説明	386
名前	386
概要	386
値	387
説明	387
名前	387
概要	387
値	387
説明	388
名前	388
概要	388
値	388
説明	388
名前	389
概要	389
値	389
説明	389
名前	390
概要	390
値	390
説明	390
名前	390
概要	390
値	390
名前	391
概要	391
値	391
説明	391
名前	391
概要	391
値	391
説明	392
名前	392

概要	392
値	392
名前	393
概要	393
値	393
名前	393
概要	393
値	393
名前	394
概要	394
値	394
名前	395
概要	395
値	395
名前	395
概要	395
値	396
名前	396
概要	396
値	396
名前	397
概要	397
値	397
名前	398
概要	398
値	398
名前	399
概要	399
値	399
名前	400
概要	400
値	400
名前	400
概要	400
値	401
名前	401
概要	401
値	401
名前	402
概要	402
値	402
<b>第33章 予測 .....</b>	<b>403</b>
名前	403
概要	403
引数	403
説明	403
名前	403
概要	403
引数	403
名前	403
概要	403
引数	403

説明	404
名前	404
概要	404
引数	404
説明	404
<b>第34章 JSON タップセット .....</b>	<b>405</b>
名前	405
概要	405
引数	405
説明	405
名前	405
概要	405
引数	405
説明	406
名前	406
概要	406
引数	406
説明	406
名前	406
概要	406
引数	406
説明	407
名前	407
概要	407
引数	407
説明	407
名前	407
概要	407
引数	407
説明	407
名前	407
概要	407
引数	407
説明	407
名前	407
概要	408
引数	408
説明	408
名前	408
概要	408
引数	408
説明	409
名前	409
概要	409
引数	409
説明	409
名前	409
概要	409
引数	409
説明	409
名前	409
概要	409
引数	409
説明	410
名前	410
概要	410

引数	410
説明	410
名前	410
概要	410
値	410
コンテキスト	410
<b>第35章 出力ファイル切り替えタップセット .....</b>	<b>412</b>
名前	412
概要	412
引数	412
説明	412
<b>付録A 改訂履歴 .....</b>	<b>413</b>



## 第1章 はじめに

**SystemTap** は、稼働中の **Linux** システムに関する情報の収集を単純化する無料のソフトウェア (**GPL**) インフラストラクチャーを提供します。これにより、パフォーマンスまたは機能の問題の診断が支援されます。また、開発者は、データを収集するのに必要になることがある退屈で破壊的なインストールメンテーション、再コンパイル、インストール、および再起動を行う必要がなくなります。

**SystemTap** は、稼働中のカーネルに対してインストールメンテーションを記述するために簡単なコマンドラインインターフェースとスクリプト言語を提供します。このインストールメンテーションは、タップセットライブラリーで提供されるプローブポイントと関数を使用します。

要するに、タップセットは、カーネルサブシステムに関する知識を他のスクリプトが使用できる事前記述されたプローブおよび関数へカプセル化するスクリプトです。タップセットは **C** プログラムのライブラリーと似ています。カーネルエリアの基礎になる詳細を公開せずに、カーネルの管理および監視に必要な主要情報を公開します。通常、タップセットはカーネルの専門家によって開発されます。

タップセットは高レベルなデータとサブシステムの状態遷移を公開します。通常、タップセットの開発者は、**SystemTap** ユーザーがカーネルサブシステムの低レベルな詳細をほとんど知らないと仮定します。したがって、タップセットの開発者は通常の **SystemTap** ユーザーが有用で便利な **SystemTap** スクリプトを書けるようにするタップセットを作成します。

### 1.1. 本ガイドの目的

本ガイドは、**SystemTap** の最も便利で一般的なタップセットを取り上げることが目的としています。また、本ガイドには、適切なタップセットの開発や文書化に関するガイドラインも含まれています。本ガイドに含まれるタップセット定義は、各タップセットファイルのコードにある正しい書式のコメントから自動的に抽出されました。そのため、本ガイドの定義への訂正は対応するタップセットファイルに直接適用する必要があります。



## 第2章 タップセット開発ガイドライン

本章では、タップセットを適切に記録するアップストリームのガイドラインについて説明します。また、本ガイドで適切に定義されるよう、適切にタップセットを記録する方法も取り上げます。

### 2.1. タップセットの記述

適切なタップセットを記述するための最初のステップとして、主題の簡単なモデルを作成します。たとえば、プロセスサブシステムのモデルには以下が含まれます。

#### キーデータ

- プロセス ID
- 親プロセス ID
- プロセスグループ ID

#### 状態遷移

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



#### 注記

上記のリストは例であり、完全なリストではありません。

サブシステムの知識を活用して、モデルの要素を公開するプローブポイント (関数エントリーおよび終了) を見つけた後、これらのポイントのプローブエイリアスを定義します。複数の場所で発生する状態遷移があることに注意してください。このような場合、エイリアスはプローブを複数の場所に置くことができます。

たとえば、プロセス `exec` は `do_execve()` または `compat_do_execve()` 関数で発生させることができます。以下のエイリアスは、これらの関数の先頭にプローブを挿入します。

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

プローブは可能な限り安定したインターフェースに置いてください (インターフェースレベルで変更されない関数など)。これにより、カーネルの変更によってタップセットが破損する可能性が低くなります。カーネルバージョンやアーキテクチャーの依存関係を回避できない場合、プリプロセッサの条件を使用します (詳細は `stap(1)` の `man` ページを参照してください)。

プローブポイントのキーデータをプローブボディに追加します。関数エントリープローブは、関数に指定したエントリーパラメーターにアクセスできます。終了プローブは、エントリーパラメーターと戻り値にアクセスできます。適切な場合はデータを意味のある形式に変換します (バイトからキロバイ

ト、状態値から文字列など)。

補助関数を使用して一部のデータにアクセスしたり、変換したりする必要がある場合があります。通常、補助関数は埋め込み C を使用して、コンテキストの構造フィールドへのアクセスやリストのリンク先への移動など、**SystemTap** 言語で実行できないことを行います。他のタップセットに定義された補助関数を使用したり、独自の補助関数を作成したりすることができます。

以下の例では、**copy\_process()** は新しいプロセスの **task\_struct** へのポインターを返します。**task\_pid()** を呼び出し、**task\_struct** ポインターへ渡すことで新しいプロセスのプロセス ID が取得されます。この場合、補助関数は **task.stp** に定義される埋め込み C 関数になります。

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

すべての関数のプローブを作成することは推奨されません。ほとんどの **SystemTap** ユーザーはそれらを必要としたり、理解しておく必要はありません。簡単に高レベルなタップセットを作成するようにしてください。

## 2.2. タップセットの要素

以降の項では、タップセットの作成で最も重要となる事柄について説明します。内容はほぼ **SystemTap** のタップセットのアップストリームライブラリーへの貢献を検討している開発者向けになります。

### 2.2.1. タップセットファイル

タップセットファイルは **SystemTap** GIT ディレクトリーの **src/tapset/** に保存されます。ほとんどのタップセットファイルがこのレベルで保存されます。特定のアーキテクチャーやカーネルバージョンでのみ動作するコードがある場合、タップセットを適切なサブディレクトリーに置くことを選択することもできます。

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

個人のタップセットはどこにでも保存できますが、**SystemTap** でこれらを使用できるようにするには、**-I tapset\_directory** を使用して **stap** を呼び出すときにタップセットの場所を指定します。

### 2.2.2. 名前空間

プローブエイリアス名は、**tapset\_name.probe\_name** という形式を取る必要があります。たとえば、シグナル送信のプローブには **signal.send** という名前を付けることができます。

グローバルシンボル名 (プローブ、関数、および変数) はタップセット全体で一意である必要があります。これにより、複数のタップセットを使用するスクリプトで名前空間の競合が発生しないようにします。このために、グローバルシンボルにタップセット固有の接頭辞を使用します。

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

### 2.2.3. コメントおよび記録

すべてのプローブと関数には、目的、提供されるデータ、および実行されるコンテキスト (割り込み、プロセスなど) を記述するコメントブロックが含まれる必要があります。コードを見ても目的が分かりにくい部分にコメントを使用します。

本ガイドには、ほとんどのタップセットから自動的に抽出された特別な形式のコメントが含まれています。これにより、タップセットの提供者が同じ場所でタップセットを作成し、記録することができます。タップセットを記録するための指定の形式は次のとおりです。

```
/**
 * probe tapset.name - タップセットの簡単な説明
 * @argument: 引数の説明。
 * @argument2: 引数2 の説明。プローブには複数の引数を使用できます。
 *
 * Context:
 * タップセットコンテキストの簡単な説明。
 * コンテキストの長さは 1 パラグラフのみ。
 *
 * 「Description」の下に表示されるテキスト。
 *
 * 「Description」の下に表示される新しいパラグラフ。
 * Header:
 * 「Header」の下に表示されるパラグラフ。
 */
```

例

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

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

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

例

```
/**
 * probe signal.handle - Fires when the signal handler is invoked
 * @sig: The signal number that invoked the signal handler
 *
 * Synopsis:
 * <programlisting>static int handle_signal(unsigned long sig, siginfo_t
```

```
*info, struct k_sigaction *ka,  
 * sigset_t *oldset, struct pt_regs * regs)</programlisting>  
*/
```

エントリーの **Synopsis** コンテンツをオーバーライドすると必要なタグが自動的に作成されないため、この例では **<programlisting>** タグを使用することが推奨されます。

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

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

## 第3章 コンテキスト関数

コンテキスト関数は、イベントが発生した場所の追加情報を提供します。これらの関数は、イベントが発生した場所へのバックトレースやプロセッサの現在のレジスター値などの情報を提供します。

### 名前

**function::addr** – 現在のプローブポイントのアドレス。

### 概要

```
addr:long()
```

### 引数

なし

### 説明

現在のプローブのレジスター状態から命令ポインターを返します。ただし、すべてのプローブタイプがレジスターを持つとは限りません。その場合は、ゼロが返されます。返されるアドレスは、**symname** や **syndata** などの関数での使用に適しています。

### 名前

**function::asmlinkage** – 宣言された **asmlinkage** として関数をマークします。

### 概要

```
asmlinkage()
```

### 引数

なし

### 説明

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

### 名前

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

### 概要

```
backtrace:string()
```

### 引数

なし

### 説明

この関数は、カーネルスタックのバックトレースである 16 進アドレスの文字列を返します。出力は文字列の最大長 (**MAXSTRINGLEN**) に従って切り捨てられることがあります。ユーザー空間バックトレースについては、**ubacktrace** を参照してください。

---

## 名前

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

## 概要

```
caller:string()
```

## 引数

なし

## 説明

この関数は、呼び出し元関数のアドレスおよび名前を返します。これは、**calling: sprintf(「s 0xx」, symname(caller\_addr), caller\_addr)** と同等です。

---

## 名前

**function::caller\_addr** – 呼び出し元アドレスを返します。

## 概要

```
caller_addr:long()
```

## 引数

なし

## 説明

この関数は呼び出し元関数のアドレスを返します。

---

## 名前

**function::callers** – カーネルスタックバックトレースの最初の **n** 要素を返します。

## 概要

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

## 引数

**n**

スタックで下がるレベルの数 (最上位レベルはカウントしません)。n が -1 の場合は、スタック全体を出力します。

## 説明

この関数は、カーネルスタックのバックトレースから最初の **n16** 進アドレスの文字列を返します。出力は、文字列の最大長 (**MAXSTRINGLEN**) に従って切り捨てられることがあります。

---

## 名前

**function::cmdline\_arg** – コマンドライン引数を取得します。

## 概要

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

## 引数

**n**

取得する引数 (**0** (ゼロ) はプログラム自体)。

## 説明

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

---

## 名前

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

## 概要

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

## 引数

**n**

最初に取得する引数 (通常、**0** (ゼロ) はプログラム自体)。

**m**

最後に取得する引数 (**-1** は **n** の後の引数すべて)。

**delim**

複数の引数を区切るために使用する文字列。

## 説明

現在のプロセスから引数番号 **n** から **m** までの引数を返します。引数の数が **n** よりも少ない場合や、現在のプロセスから引数を取得できない場合は、空の文字列が返されます。**m** が **n** よりも小さい場合は、引数 **n** 以降のすべての引数が返されます。通常、引数 **0** はコマンド自体になります。

---

## 名前

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

## 概要

```
cmdline_str:string()
```

## 引数

なし

## 説明

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

---

## 名前

**function::cpu** – 現在の CPU 番号を返します。

## 概要

```
cpu:long()
```

## 引数

なし

## 説明

この関数は、現在の CPU 番号を返します。

---

## 名前

**function::cpuid** – 現在の CPU 番号を返します。

## 概要

```
cpuid:long()
```

## 引数

なし

## 説明

この関数は、現在の CPU 番号を返します。SystemTap 1.4 で非推奨になり、SystemTap 1.5 で削除されます。

---

## 名前

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



## 概要

```
egid:long()
```

## 引数

なし

## 説明

この関数は、ターゲットプロセスの実効 **GID** を返します。

---

## 名前

**function::env\_var** – 現在のプロセスから環境変数を取得します。

## 概要

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

## 引数

### **name**

取得する環境変数の名前。

## 説明

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

---

## 名前

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

## 概要

```
euid:long()
```

## 引数

なし

## 説明

ターゲットプロセスの実効ユーザー **ID** を返します。

---

## 名前

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

## 概要

```
execname:string()
```

## 引数

なし

## 説明

ターゲットプロセス (またはプロセスのグループ) の実行名を返します。

---

## 名前

**function::fastcall** – 宣言された **fastcall** として関数をマークします。

## 概要

```
fastcall()
```

## 引数

なし

## 説明

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

---

## 名前

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

## 概要

```
gid:long()
```

## 引数

なし

## 説明

この関数は、ターゲットプロセスのグループ ID を返します。

---

## 名前

**function::int\_arg** – 関数の引数を符号付き **int** として返します。

## 概要

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

## 引数

***n***

返す引数のインデックス。

## 説明

引数 **n** の値を符号付き **int** として返します (64 ビットに符号拡張された 32 ビット整数など)。

---

## 名前

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

## 概要

```
is_myproc:long()
```

## 引数

なし

## 説明

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

---

## 名前

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

## 概要

```
is_return:long()
```

## 引数

なし

## 説明

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

---

## 名前

**function::long\_arg** – 関数の引数を符号付き **long** として返します。

## 概要

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

## 引数

***n***

返す引数のインデックス。

## 説明

引数 *n* の値を符号付き **long** として返します。**long** が 32 ビットであるアーキテクチャーの場合、値は 64 ビットに符号拡張されます。

---

## 名前

**function::longlong\_arg** – 関数の引数を 64 ビット値として返します。

## 概要

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

## 引数

***n***

返す引数のインデックス。

## 説明

引数 *n* の値を 64 ビット値として返します。

---

## 名前

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

## 概要

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

## 引数

***addr***

カーネルモジュール名にマップされるアドレス。

## 説明

既知の場合は、指定のアドレスに関連するモジュール名を返します。不明な場合は、エラーを発生させます。アドレスがカーネルモジュールではなく、カーネル自体にあった場合は、文字列「**kernel**」を返します。

---

## 名前

**function::module\_name** – 現在のスクリプトのモジュール名。

## 概要

```
module_name:string()
```

## 引数

なし

## 説明

この関数は、**stap** モジュールの名前を返します。無作為に生成されるか (**stap\_[0-9a-f]\_[0-9a-f]+**)、**stap -m <module\_name>** によって設定されます。

---

## 名前

**function::module\_size** – 現在のスクリプトのモジュールサイズ。

## 概要

```
module_size:string()
```

## 引数

なし

## 説明

この関数は、**stap** モジュールのさまざまなセクションのサイズを返します。

---

## 名前

**function::ns\_egid** – ユーザーネームスペースで示されたターゲットプロセスの実効 GID を返します。

## 概要

```
ns_egid:long()
```

## 引数

なし

## 説明

この関数は、ターゲットユーザーネームスペース (提供された場合) または **stap** プロセスネームスペースで示されたターゲットプロセスの実効 GID を返します。

---

## 名前

**function::ns\_euid** – ユーザーネームスペースで示されたターゲットプロセスの実効ユーザー ID を返します。

## 概要

```
ns_euid:long()
```

## 引数

なし

## 説明

この関数は、ターゲットユーザーネームスペース (提供された場合) または **stap** プロセスネームスペースで示されたターゲットプロセスの実効ユーザー ID を返します。

---

## 名前

**function::ns\_gid** – ユーザーネームスペースで示されたターゲットプロセスのグループ ID を返します。

## 概要

```
ns_gid:long()
```

## 引数

なし

## 説明

この関数は、ターゲットユーザーネームスペース (提供された場合) または **stap** プロセスネームスペースで示されたターゲットプロセスのグループ ID を返します。

---

## 名前

**function::ns\_pgrp – pid** ネームスペースで示された現在のプロセスのプロセスグループ ID を返します。

## 概要

```
ns_pgrp:long()
```

## 引数

なし

## 説明

この関数は、ターゲット **pid** ネームスペース (提供された場合) または **stap** プロセスネームスペースで示された現在のプロセスのプロセスグループ ID を返します。

---

## 名前

**function::ns\_pid – pid** ネームスペースで示されたターゲットプロセスの ID を返します。

## 概要

```
ns_pid:long()
```

## 引数

なし

## 説明

この関数は、ターゲット **pid** ネームスペースで示されたターゲットプロセスの **ID** を返します。

---

## 名前

**function::ns\_ppid** – **pid** ネームスペースで示されたターゲットプロセスの親プロセスのプロセス **ID** を返します。

## 概要

```
ns_ppid:long()
```

## 引数

なし

## 説明

この関数は、ターゲット **pid** ネームスペース (提供された場合) または **stap** プロセスネームスペースで示されたターゲットプロセスの親プロセスのプロセス **ID** を返します。

---

## 名前

**function::ns\_sid** – **pid** ネームスペースで示された現在のプロセスのセッション **ID** を返します。

## 概要

```
ns_sid:long()
```

## 引数

なし

## 説明

プロセスのネームスペース対応セッション **ID** は、ターゲット **pid** ネームスペース (提供された場合) または **stap** プロセスネームスペースで示されたセッションリーダーのプロセスグループ **ID** です。カーネル 2.6.0 以降、セッション **ID** は **signal\_struct** に保存されます。

---

## 名前

**function::ns\_tid** – **pid** ネームスペースで示されたターゲットプロセスのスレッド **ID** を返します。

## 概要

```
ns_tid:long()
```

## 引数

なし

## 説明

この関数は、ターゲット **pid** ネームスペース (提供された場合) または **stap** プロセスネームスペースで示されたターゲットプロセスのスレッド ID を返します。

---

## 名前

**function::ns\_uid** – ユーザーネームスペースで示されたターゲットプロセスのユーザー ID を返します。

## 概要

```
ns_uid:long()
```

## 引数

なし

## 説明

この関数は、ターゲットユーザーネームスペース (提供された場合) または **stap** プロセスネームスペースで示されたターゲットプロセスのユーザー ID を返します。

---

## 名前

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

## 概要

```
pexecname:string()
```

## 引数

なし

## 説明

この関数は、ターゲットプロセスの親プロセスの実行名を返します。

---

## 名前

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

## 概要

```
pgrp:long()
```

## 引数

なし

## 説明



この関数は、現在のプロセスのプロセスグループ ID を返します。

---

## 名前

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

## 概要

```
pid:long()
```

## 引数

なし

## 説明

この関数は、ターゲットプロセスの ID を返します。

---

## 名前

**function::pid2execname** – 指定のプロセス ID の名前。

## 概要

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

## 引数

### ***pid***

プロセス ID。

## 説明

指定のプロセス ID の名前を返します。

---

## 名前

**function::pid2task** – 指定のプロセス ID の task\_struct。

## 概要

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

## 引数

### ***pid***

プロセス ID。

## 説明

指定のプロセス ID の **task** 構造を返します。

---

## 名前

**function::pn** – アクティブなプローブ名を返します。

## 概要

```
pn:string()
```

## 引数

なし

## 説明

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

---

## 名前

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

## 概要

```
pnlabel:string()
```

## 引数

なし

## 説明

スクリプトレベルプローブポイントから解析されたラベル名を返します。この関数は、**!.label** プローブポイントから直接呼び出された場合 (つまり、エイリアスなし) のみ動作します。

## コンテキスト

現在のプローブポイント。

---

## 名前

**function::pointer\_arg** – 関数の引数をポインター値として返します。

## 概要

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

## 引数

*n*

返す引数のインデックス。

## 説明

`ulong_arg` と同様に、引数 `n` の符号なし値を返します。どのタイプのポインターでも使用できます。

---

## 名前

`function::pp` – アクティブなプローブポイントを返します。

## 概要

```
pp:string()
```

## 引数

なし

## 説明

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

---

## 名前

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

## 概要

```
ppfunc:string()
```

## 引数

なし

## 説明

現在の `pp` から関数名を返します。すべての `pp` が関数を持つとは限りません。関数を持たない場合は、`""` が返されます。

---

## 名前

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

## 概要

```
ppid:long()
```

## 引数

なし

## 説明

この関数は、ターゲットプロセスの親プロセスのプロセス ID を返します。

---

## 名前

**function::print\_backtrace** – カーネルスタックバックトレースを出力します。

## 概要

```
print_backtrace()
```

## 引数

なし

## 説明

この関数は **print\_stack(backtrace)** と同等ですが、より深度の高いスタックのネストがサポートされます。ユーザー空間バックトレースについては、**print\_ubacktrace** を参照してください。この関数は値を返しません。

---

## 名前

**function::print\_regs** – レジスタダンプを出力します。

## 概要

```
print_regs()
```

## 引数

なし

## 説明

この関数はレジスタダンプを出力します。プローブポイントに利用可能なレジスタがない場合は、何も行いません。

---

## 名前

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

## 概要

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

## 引数

**stk**

16 進アドレスのリストが含まれる文字列。

## 説明

この関数は、**backtrace**への事前呼び出しの結果となることが仮定される指定の文字列でアドレスのシンボリックルックアップを実行します。

アドレスごとに1行出力し、アドレス、アドレスが含まれる関数の名前、およびその関数内での推定位置が含まれます。戻り値はありません。

## 注記

この関数の代わりに **print\_syms** を使用することが推奨されます。

---

## 名前

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

## 概要

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

## 引数

### **callers**

16 進 (カーネル) アドレスのリストが含まれる文字列。

## 説明

この関数は、**stack**、**callers**、および類似の関数への事前呼び出しの結果となることが仮定される指定の文字列でアドレスのシンボリックルックアップを実行します。

**symdata** により取得されるように、アドレスごとに1行出力します。アドレス、アドレスが含まれる関数の名前、およびその関数内での推定位置が含まれます。戻り値はありません。

---

## 名前

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

## 概要

```
print_ubacktrace()
```

## 引数

なし

## 説明

この関数は **print\_ustack(ubacktrace)** と同等ですが、より深度の高いスタックのネストがサポートされます。この関数は値を返しません。カーネルバックトレースについては、**print\_backtrace** を参照してください。

## 注記

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

---

## 名前

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

## 概要

```
print_ubacktrace_brief()
```

## 引数

なし

## 説明

**print\_ubacktrace** と同等ですが、各シンボルの出力が簡素化されます (名前とオフセットのみ、またはシンボルが見つからない場合は 16 進アドレスのみ)。

## 注記

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

---

## 名前

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

## 概要

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

## 引数

**stk**

現在のタスクの 16 進アドレスのリストが含まれる文字列。

## 説明

現在のタスクの **ubacktrace** への事前呼び出しの結果となることが仮定される指定の文字列でアドレスのシンボリックルックアップを実行します。

アドレスごとに 1 行出力し、アドレス、アドレスが含まれる関数の名前、およびその関数内での推定位置が含まれます。戻り値はありません。

## 注記

この関数の代わりに **print\_usyms** を使用することが推奨されます。

---

## 名前

**function::print\_usyms** – 文字列からユーザースタックを出力します。

## 概要

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

## 引数

### **callers**

16 進 (ユーザー) アドレスのリストが含まれる文字列。

## 説明

この関数は、**ustack**、**ucallers**、および類似の関数への事前呼び出しの結果となることが仮定される指定の文字列でアドレスのシンボリックルックアップを実行します。

**usymdata** により取得されるように、アドレスごとに 1 行出力します。アドレス、アドレスが含まれる関数の名前、およびその関数内での推定位置が含まれます。戻り値はありません。

## 名前

**function::probe\_type** – 現在のプローブの低レベルプローブハンドラー。

## 概要

```
probe_type:string()
```

## 引数

なし

## 説明

現在のプローブポイントの低レベルプローブハンドラーを説明する短い文字列を返します。これは、情報提供のみを目的とします。低レベルプローブハンドラーに応じて、異なるコンテキスト関数が現在のイベントに関する情報を提供できたり、できなかったりします (たとえば、一部のプローブハンドラーは、ユーザー空間でのみトリガーされ、関連するカーネルコンテキストを持ちません)。高レベルプローブは、同じ、または異なる低レベルプローブにマップできることがあります (使用している **systemtap** バージョンとカーネルに依存します)。

## 名前

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

## 概要

```
probefunc:string()
```

## 引数

なし

## 説明

この関数は、現在のアドレスに基づいてプローブされる関数の名前を返します。プローブコンテキストに応じて (プローブがユーザープローブであるか、またはカーネルプローブであるか)、**symname(addr)** または **usymname(uaddr)** によって計算されます。

## 注記

この関数の動作は、**SystemTap 2.0** とそれより前のバージョンでは異なります。**2.0** より前のバージョンでは、**probfunc** は **pp** により返されたプローブポイント文字列から関数名を取得し、現在のアドレスをフォールバックとして使用していました。

代わりに **ppfunc** を使用することを検討してください。

---

## 名前

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

## 概要

```
probemod:string()
```

## 引数

なし

## 説明

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

---

## 名前

**function::pstrace** – プロセスのチェーンと **init(1)** までの **pid**。

## 概要

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

## 引数

### **task**

プロセスの **task** 構造へのポインター。

## 説明

この関数は、**task** から **init(1)** が生成した親プロセスまでの各プロセスの実行名と **pid** をリストする文字列を返します。

---

## 名前

**function::register** – 名前付き CPU レジスタの符号付き値を返します。



## 概要

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

## 引数

### *name*

返すレジスタの名前。

## 説明

現在のプローブポイントがヒットしたときに保存された名前付き CPU レジスタの値を返します。レジスタが 32 ビットの場合は、64 ビットに符号拡張されます。

i386 アーキテクチャの場合は、次の名前が認識されます (name1/name2 は name1 と name2 が同じレジスタの代替名であることを示しています): `eax/ax`、`ebp/bp`、`ebx/bx`、`ecx/cx`、`edi/di`、`edx/dx`、`eflags/flags`、`eip/ip`、`esi/si`、`esp/sp`、`orig_eax/orig_ax`、`xcs/cs`、`xds/ds`、`xes/es`、`xfs/fs`、`xss/ss`。

x86\_64 アーキテクチャの場合は、次の名前が認識されます: 64 ビットレジスタ: `r8`、`r9`、`r10`、`r11`、`r12`、`r13`、`r14`、`r15`、`rax/ax`、`rbp/bp`、`rbx/bx`、`rcx/cx`、`rdi/di`、`rdx/dx`、`rip/ip`、`rsi/si`、`rsp/sp`; 32 ビットレジスタ: `eax`、`ebp`、`ebx`、`ecx`、`edx`、`edi`、`edx`、`eip`、`esi`、`esp`、`flags/eflags`、`orig_eax`; セグメントレジスタ: `xcs/cs`、`xss/ss`。

powerpc の場合は、次の名前が認識されます: `r0`、`r1`、... `r31`、`nip`、`msr`、`orig_gpr3`、`ctr`、`link`、`xer`、`ccr`、`softe`、`trap`、`dar`、`dsisr`、`result`。

s390x の場合は、次の名前が認識されます: `r0`、`r1`、... `r15`、`args`、`psw.mask`、`psw.addr`、`orig_gpr2`、`ilc`、`trap`。

AArch64 の場合は、次の名前が認識されます: `x0`、`x1`、... `x30`、`fp`、`lr`、`sp`、`pc`、および `orig_x0`。

## 名前

`function::registers_valid` – 現在のコンテキストにおける `register` および `u_register` の有効性を決定します。

## 概要

```
registers_valid:long()
```

## 引数

なし

## 説明

`register` および `u_register` を現在のコンテキストで 사용할 ことができる場合、この関数は 1 を返します。使用できない場合は 0 を返します。たとえば、`begin` または `end` プローブから呼び出されると `registers_valid` は 0 を返します。

## 名前

**function::regparm** – 関数をコンパイルするのに使用する **regparm** 値を指定します。

## 概要

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

## 引数

*n*

元の **regparm** 値。

## 説明

関数が **gcc -mregparm=n** でビルドされた場合に、**\*\_arg** 関数を使用して関数引数にアクセスする前にこの関数を呼び出します。

(i386 カーネルは **\-mregparm=3** でビルドされます。従って、**systemtap** はこのアーキテクチャーでは **regparm(3)** をカーネル関数のデフォルトと見なします)。i386 と x86\_64 でのみ有効です (32 ビットアプリケーションをプローブする場合)。他のアーキテクチャーでは、エラーを発生させます。

---

## 名前

**function::remote\_id** – リモート実行でのこのインスタンスのインデックス。

## 概要

```
remote_id:long()
```

## 引数

なし

## 説明

この関数は番号 0..N を返します。これは一連の「**stap --remote A --remote B ...**」実行からの特定スクリプト実行の固有インデックスであり、「**stap --remote-prefix**」が出力する番号と同じです。この関数は、スクリプトが「**stap --remote**」で起動されなかった場合や、リモート **staprun/stapsh** がバージョン 1.7 よりも古い場合に -1 を返します。

---

## 名前

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

## 概要

```
remote_uri:string()
```

## 引数

なし

## 説明

この関数は、一連の「**stap --remote**」実行からのこの特定スクリプト実行を起動するために使用されるリモートホストを返します。これは一意でない場合があります。この関数は、スクリプトが「**stap --remote**」で起動されなかった場合に空の文字列を返します。

---

## 名前

**function::s32\_arg** – 関数の引数を符号付き 32 ビット値として返します。

## 概要

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

## 引数

*n*

返す引数のインデックス。

## 説明

**int\_arg** と同様に、引数 *n* の符号付き 32 ビット値を返します。

---

## 名前

**function::s64\_arg** – 関数の引数を符号付き 64 ビット値として返します。

## 概要

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

## 引数

*n*

返す引数のインデックス。

## 説明

**longlong\_arg** と同様に、引数の符号付き 64 ビット値を返します。

---

## 名前

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

## 概要

```
sid:long()
```

## 引数

なし

## 説明

プロセスのセッション ID は、セッションリーダーのプロセスグループ ID です。カーネル 2.6.0 以降、セッション ID は `signal_struct` に保存されます。

---

## 名前

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

## 概要

```
sprint_backtrace:string()
```

## 引数

なし

## 説明

簡単な (カーネル) バックトレースを返します。アドレスごとに 1 行使用します。シンボル名 (またはシンボルを解決できなかった場合は 16 進アドレス) およびモジュール名 (見つかった場合) が含まれます。見つかった場合は関数の始めからのオフセットが含まれ、見つからなかった場合はオフセットはモジュールに追加されます (見つかった場合カッコで囲む)。バックトレースを文字列として返します (各行は改行文字で終わります)。返されるスタックは `MAXSTRINGLEN` に切り捨てられます。完全なスタックを出力するには `print_backtrace` を使用します。`sprint_stack(backtrace)` と同等ですが、効率がよくなります (16 進文字列と最終のバックトレース文字列の変換を実行する必要がありません)。

---

## 名前

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

## 概要

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

## 引数

`stk`

16 進 (カーネル) アドレスのリストが含まれる文字列。

## 説明

`backtrace` への事前呼び出しの結果となることが仮定される指定の文字列でアドレスのシンボリックルックアップを実行します。

指定の 16 進文字列から簡単なバックトレースを返します。アドレスごとに 1 行が返されます。これには、シンボル名 (またはシンボルを解決できなかった場合は 16 進アドレス) およびモジュール名 (見つかった場合) が含まれます。見つかった場合は関数の始めからオフセットが組み込まれ、見つからなかった場合はオフセットがモジュールに追加されます (見つかった場合はカッコで囲む)。バックトレースを文字列として返します (各行は改行文字で終わる)。返されるスタックは `MAXSTRINGLEN` に切り捨てられます。完全にリッチなスタックを出力するには `print_stack` を使用します。

## 注記

この関数の代わりに **sprint\_syms** を使用することが推奨されます。

## 名前

**function::sprint\_syms** – 文字列からカーネルアドレスのスタックを返します。

## 概要

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

## 引数

### **callers**

16 進 (カーネル) アドレスのリストが含まれる文字列。

## 説明

**stack**、**callers**、および類似の関数への事前呼び出しの結果となることが仮定される指定の文字列でアドレスのシンボリックルックアップを実行します。

指定の 16 進文字列から簡単なバックトレースを返します。アドレスごとに 1 行使用します。 **syndata** から取得されたシンボル名 (またはシンボルを解決できなかった場合は 16 進アドレス) およびモジュール名 (見つかった場合) が含まれます。見つかった場合は関数の始めからのオフセットが含まれ、見つからなかった場合はオフセットはモジュールに追加されます (見つかった場合はかっこで囲む)。バックトレースを文字列として返します (各行は改行文字で終わる)。返されるスタックは **MAXSTRINGLEN** に切り捨てられます。完全でリッチなスタックを出力するには **print\_syms** を使用します。

## 名前

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

## 概要

```
sprint_ubacktrace:string()
```

## 引数

なし

## 説明

現在のタスクの簡単なバックトレースを返します。アドレスごとに 1 行使用します。シンボル名 (またはシンボルを解決できなかった場合は 16 進アドレス) およびモジュール名 (見つかった場合) が含まれます。見つかった場合は関数の始めからのオフセットが含まれ、見つからなかった場合はオフセットはモジュールに追加されます (見つかった場合はかっこで囲む)。バックトレースを文字列として返します (各行は改行文字で終わる)。返されるスタックは **MAXSTRINGLEN** に切り捨てられます。完全でリッチなスタックを出力するには **print\_ubacktrace** を使用します。 **sprint\_ustack(ubacktrace)** と同等ですが、効率がよくなります (16 進文字列と最終のバックトレース文字列の変換を実行する必要がありません)。

## 注記

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

---

## 名前

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

## 概要

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

## 引数

### *stk*

現在のタスクの 16 進アドレスのリストが含まれる文字列。

## 説明

現在のタスクの **ubacktrace** への事前呼び出しの結果となることが仮定される指定の文字列でアドレスのシンボリックルックアップを実行します。

指定の 16 進文字列から簡単なバックトレースを返します。アドレスごとに 1 行使用します。シンボル名 (またはシンボルを解決できなかった場合は 16 進アドレス) およびモジュール名 (見つかった場合) が含まれます。見つかった場合は関数の始めからのオフセットが含まれ、見つからなかった場合はオフセットはモジュールに追加されます (見つかった場合はかっこで囲む)。バックトレースを文字列として返します (各行は改行文字で終わる)。返されるスタックは **MAXSTRINGLEN** に切り捨てられます。完全でリッチなスタックを出力するには **print\_ustack** を使用します。

## 注記

この関数の代わりに **sprint\_usyms** を使用することが推奨されます。

---

## 名前

**function::sprint\_usyms** – 文字列からユーザーアドレスのスタックを返します。

## 概要

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

## 引数

### *callers*

16 進 (ユーザー) アドレスのリストが含まれる文字列。

## 説明

**ustack**、**ucallers**、および類似の関数への事前呼び出しの結果となることが仮定される指定の文字列でアドレスのシンボリックルックアップを実行します。

指定の16進文字列から簡単なバックトレースを返します。アドレスごとに1行使用します。**usymdata**から取得されたシンボル名(またはシンボルを解決できなかった場合は16進アドレス)およびモジュール名(見つかった場合)が含まれます。見つかった場合は関数の始めからのオフセットが含まれ、見つからなかった場合はオフセットはモジュールに追加されます(見つかった場合はかっこで囲む)。バックトレースを文字列として返します(各行は改行文字で終わる)。返されるスタックはMAXSTRINGLENに切り捨てられます。完全でリッチなスタックを出力するには**print\_usyms**を使用します。

## 名前

**function::stack** – カーネルスタックバックトレースの指定の深さでアドレスを返します。

## 概要

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

## 引数

*n*

スタックで下がるレベルの数。

## 説明

単純な(カーネル)バックトレースを実行し、指定された位置の要素を返します。バックトレース自体の結果はキャッシュされ、**stack**が呼び出される回数や順序に関係なく、バックトレースの計算は最大で1回実行されます。

## 名前

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

## 概要

```
stack_size:long()
```

## 引数

なし

## 説明

この関数は、カーネルスタックのサイズを返します。

## 名前

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

## 概要

```
stack_unused:long()
```

## 引数

なし

## 説明

この関数は、カーネルスタックの現在の空き容量 (バイト数) を判断します。

---

## 名前

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

## 概要

```
stack_used:long()
```

## 引数

なし

## 説明

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

---

## 名前

**function::stp\_pid** – stapio プロセスのプロセス ID。

## 概要

```
stp_pid:long()
```

## 引数

なし

## 説明

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

---

## 名前

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

## 概要

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



## 引数

### ***addr***

変換するアドレス。

## 説明

既知の場合、指定のアドレスに関連する (関数) シンボル名を返します。始めからのオフセット、シンボルのサイズ、およびモジュール名 (かっこで囲まれる) も返します。シンボルが不明でモジュールは既知の場合、モジュール内のオフセットとモジュールのサイズが追加されます。要素が不明な場合は省略され、シンボル名が不明な場合は指定のアドレスの16進文字列を返します。

---

## 名前

**function::symfile** – 指定アドレスのファイル名を返します。

## 概要

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

## 引数

### ***addr***

変換するアドレス。

## 説明

既知の場合、指定のアドレスのファイル名を返します。ファイル名が不明な場合は、アドレスの16進文字列表記を返します。

---

## 名前

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

## 概要

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

## 引数

### ***addr***

変換するアドレス。

## 説明

既知の場合、指定のアドレスのファイル名と (おおよその) 行番号を返します。ファイル名または行番号が不明な場合は、アドレスの16進文字列表記を返します。

---

## 名前

**function::symline** – アドレスの行番号を返します。

## 概要

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

## 引数

### **addr**

変換するアドレス。

## 説明

既知の場合、指定のアドレスの (おおよその) 行番号を返します。行番号が不明な場合は、アドレスの 16 進文字列表記を返します。

---

## 名前

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

## 概要

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

## 引数

### **addr**

変換するアドレス。

## 説明

既知の場合、指定のアドレスに関連する (関数) シンボル名を返します。不明な場合は、アドレスの 16 進文字列表記を返します。

---

## 名前

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

## 概要

```
target:long()
```

## 引数

なし

## 説明

この関数は、ターゲットプロセスのプロセス ID を返します。これは、**-x PID** または **-c CMD** コマンドラインオプションとともに **stap** に使用すると便利です。たとえば、特定のプロセスでフィルターするスクリプトを作成する場合に使用できます。

**-x <pid> target** は、**-x** で指定された **pid** を返します。

**target** は、**-c** で指定された実行済みコマンドの **pid** を返します。

---

## 名前

**function::task\_ancestry** – 指定のタスクの先祖。

## 概要

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

## 引数

### **task**

**task\_struct** ポインター。

### **with\_time**

プロセスの起動時刻を出力するためにも 1 に設定されます (起動時刻からのデルタ値として指定)。

## 説明

指定のタスクの先祖を「**grandparent\_process=>parent\_process=>process**」という形式で返します。

---

## 名前

**function::task\_backtrace** – 任意タスクの 16 進バックトレース。

## 概要

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

## 引数

### **task**

**task\_struct** へのポインター。

## 説明

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

---

## 名前

**function::task\_cpu** – タスクのスケジュールされた CPU。

## 概要

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

## 引数

### *task*

**task\_struct** ポインター。

## 説明

この関数は、指定タスクのスケジュールされた CPU を返します。

---

## 名前

**function::task\_current** – 現在のタスクの現在の **task\_struct**。

## 概要

```
task_current:long()
```

## 引数

なし

## 説明

この関数は、現在のプロセスを表す **task\_struct** を返します。このアドレスをさまざまな **task\_\***() 関数に渡すとタスク固有の詳細なデータを抽出できます。

---

## 名前

**function::task\_cwd\_path** – タスクの現在の作業ディレクトリーのパス構造ポインターを取得します。

## 概要

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

## 引数

### *task*

**task\_struct** ポインター。

---

## 名前

**function::task\_egid** – タスクの実効グループ ID。

## 概要

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

## 引数

### *task*

`task_struct` ポインター。

## 説明

この関数は、指定タスクの実効グループ ID を返します。

---

## 名前

`function::task_euid` – タスクの実効ユーザー ID。

## 概要

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

## 引数

### *task*

`task_struct` ポインター。

## 説明

この関数は、指定タスクの実効ユーザー ID を返します。

---

## 名前

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

## 概要

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

## 引数

### *task*

`task_struct` ポインター。

---

## 名前

`function::task_execname` – タスクの名前。

## 概要

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

## 引数

### *task*

`task_struct` ポインター。

## 説明

指定タスクの名前を返します。

---

## 名前

`function::task_fd_lookup` – タスクの `fd` のファイル構造を取得します。

## 概要

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

## 引数

### *task*

`task_struct` ポインター。

### *fd*

ファイル記述子番号。

## 説明

タスクのファイル記述子のファイル構造ポインターを返します。

---

## 名前

`function::task_gid` – タスクのグループ ID。

## 概要

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

## 引数

### *task*

`task_struct` ポインター。

## 説明

この関数は、指定タスクのグループ ID を返します。

---

## 名前

function::task\_max\_file\_handles – タスクのオープンファイルの最大数。

## 概要

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

## 引数

### *task*

task\_struct ポインター。

## 説明

この関数は、指定タスクのファイルハンドラーの最大数を返します。

---

## 名前

function::task\_nice – タスクの nice 値。

## 概要

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

## 引数

### *task*

task\_struct ポインター。

## 説明

この関数は、指定タスクの nice 値を返します。

---

## 名前

function::task\_ns\_egid – タスクの実効グループ ID。

## 概要

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

## 引数

### *task*

`task_struct` ポインター。

## 説明

この関数は、指定タスクの実効グループ ID を返します。

---

## 名前

**function::task\_ns\_euid** – タスクの実効ユーザー ID。

## 概要

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

## 引数

### *task*

`task_struct` ポインター。

## 説明

この関数は、指定タスクの実効ユーザー ID を返します。

---

## 名前

**function::task\_ns\_gid** – ネームスペースで示されるタスクのグループ ID。

## 概要

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

## 引数

### *task*

`task_struct` ポインター。

## 説明

この関数は、指定のユーザーネームスペースで示される指定タスクのグループ ID を返します。

---

## 名前

**function::task\_ns\_pid** – タスクのプロセス ID。

## 概要

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



## 引数

### *task*

`task_struct` ポインター。

## 説明

この関数は、指定 `pid` ネームスペースに基づいて指定タスクのプロセス ID を返します。

---

## 名前

`function::task_ns_tid` – ネームスペースで示されるタスクのスレッド ID。

## 概要

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

## 引数

### *task*

`task_struct` ポインター。

## 説明

この関数は、`pid` ネームスペースで示される指定タスクのスレッド ID を返します。

---

## 名前

`function::task_ns_uid` – タスクのユーザー ID。

## 概要

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

## 引数

### *task*

`task_struct` ポインター。

## 説明

この関数は、指定タスクのユーザー ID を返します。

---

## 名前

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

## 概要

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

## 引数

### *task*

`task_struct` ポインター。

## 説明

この関数は、指定タスクのオープンファイルハンドラーの数を返します。

---

## 名前

`function::task_parent` – 親タスクの `task_struct`。

## 概要

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

## 引数

### *task*

`task_struct` ポインター。

## 説明

この関数は、指定タスクの親 `task_struct` を返します。このアドレスを `task_*`() 関数に渡すとタスク固有の詳細なデータを抽出できます。

---

## 名前

`function::task_pid` – タスクのプロセス ID。

## 概要

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

## 引数

### *task*

`task_struct` ポインター。

## 説明

この関数は、指定タスクのプロセス ID を返します。

---

## 名前

**function::task\_prio** – タスクの優先度の値。

## 概要

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

## 引数

### *task*

**task\_struct** ポインター。

## 説明

この関数は、指定タスクの優先度の値を返します。

---

## 名前

**function::task\_state** – タスクの状態。

## 概要

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

## 引数

### *task*

**task\_struct** ポインター。

## 説明

指定タスクの状態を返します (**TASK\_RUNNING** (0)、**TASK\_INTERRUPTIBLE** (1)、**TASK\_UNINTERRUPTIBLE** (2)、**TASK\_STOPPED** (4)、**TASK\_TRACED** (8)、**EXIT\_ZOMBIE** (16)、または **EXIT\_DEAD** (32) のいずれか)。

---

## 名前

**function::task\_tid** – タスクのスレッド ID。

## 概要

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

## 引数

### *task*

**task\_struct** ポインター。

## 説明

この関数は、指定タスクのスレッド ID を返します。

---

## 名前

**function::task\_uid** – タスクのユーザー ID。

## 概要

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

## 引数

### **task**

**task\_struct** ポインター。

## 説明

この関数は、指定タスクのユーザー ID を返します。

---

## 名前

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

## 概要

```
tid:long()
```

## 引数

なし

## 説明

この関数はターゲットプロセスのスレッド ID を返します。

---

## 名前

**function::u32\_arg** – 関数の引数を符号なし 32 ビット値として返します。

## 概要

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

## 引数

### **n**

返す引数のインデックス。

## 説明

`uint_arg` と同様に、引数の符号なし 32 ビット値を返します。

---

## 名前

**function::u64\_arg** – 関数の引数を符号なし 64 ビット値として返します。

## 概要

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

## 引数

***n***

返す引数のインデックス。

## 説明

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

---

## 名前

**function::u\_register** – 名前付き CPU レジスターの符号なし値を返します。

## 概要

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

## 引数

***name***

返すレジスターの名前。

## 説明

`register(name)` と同様。ただし、レジスターが 32 ビット幅の場合は、64 ビットにゼロ拡張されます。

---

## 名前

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

## 概要

```
uaddr:long()
```

## 引数

なし

## 説明

プローブが発生したときに現在のタスクが存在したユーザー空間のアドレスを返します。現在実行しているタスクがユーザー空間のスレッドでなかったり、アドレスが見つからない場合は、**0** を返します。現在のタスクがどこで **usymname** または **usymdata** と組み合わせられているかを確認するために使用できます。多くの場合、タスクはカーネルに入った **VDSO** にあります。

---

## 名前

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

## 概要

```
ubacktrace:string()
```

## 引数

なし

## 説明

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

## 注記

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

---

## 名前

**function::ucallers** – ユーザースタックバックトレースの最初の **n** 要素を返します。

## 概要

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

## 引数

**n**

スタックで下がるレベルの数 (最上位レベルはカウントしません)。 **n** が **-1** の場合は、スタック全体を出力します。

## 説明

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

## 注記

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

---

## 名前

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

## 概要

```
uid:long()
```

## 引数

なし

## 説明

この関数は、ターゲットプロセスのユーザー ID を返します。

---

## 名前

**function::uint\_arg** – 関数の引数を符号なし **int** として返します。

## 概要

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

## 引数

***n***

返す引数のインデックス。

## 説明

引数 ***n*** の値を符号なし **int** として返します (64 ビットにゼロ拡張された 32 ビット整数など)。

---

## 名前

**function::ulong\_arg** – 関数の引数を符号なし **long** として返します。

## 概要

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

## 引数

***n***

返す引数のインデックス。

## 説明

引数 **n** の値を符号なし **long** として返します。**long** が 32 ビットであるアーキテクチャーの場合、値は 64 ビットにゼロ拡張されます。

---

## 名前

**function::ulonglong\_arg** – 関数の引数を 64 ビット値として返します。

## 概要

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

## 引数

**n**

返す引数のインデックス。

## 説明

引数 **n** の値を 64 ビット値として返します (**longlong\_arg** と同様)。

---

## 名前

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

## 概要

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

## 引数

**addr**

ユーザー空間アドレス

## 説明

指定のアドレスを含む現在のタスクのユーザー空間モジュールの短い名前を返します。アドレスが (マッピングされた) モジュールではない場合、または何らかの理由でモジュールが見つからない場合は、エラーを報告します。

---

## 名前

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

## 概要

```
user_mode:long()
```



## 引数

なし

## 説明

プローブポイントがユーザーモードで発生した場合は1を返します。

---

## 名前

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

## 概要

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

## 引数

***n***

スタックで下がるレベルの数。

## 説明

単純な (ユーザー空間) バックトレースを実行し、指定された位置の要素を返します。バックトレース自体の結果はキャッシュされ、**ustack** が呼び出される回数や順序に関係なく、バックトレースの計算は最大で1回実行されます。

---

## 名前

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

## 概要

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

## 引数

***addr***

変換するアドレス。

## 説明

既知の場合、現在のタスクの指定のアドレスに関連する (関数) シンボル名を返し、先頭からのオフセット、シンボルのサイズ、およびモジュール名 (かっこで囲まれる) も返します。シンボルが不明でモジュールは既知の場合は、モジュール内のオフセットとモジュールのサイズが追加されます。要素が不明な場合は省略され、シンボル名が不明な場合は指定のアドレスの16進文字列を返します。

---

## 名前

**function::usymfile** – 指定アドレスのファイル名を返します。

## 概要

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

## 引数

### ***addr***

変換するアドレス。

## 説明

既知の場合、指定のアドレスのファイル名を返します。ファイル名が不明な場合は、アドレスの **16** 進文字列表記を返します。

---

## 名前

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

## 概要

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

## 引数

### ***addr***

変換するアドレス。

## 説明

既知の場合、指定のアドレスのファイル名と (おおよその) 行番号を返します。ファイル名または行番号が不明な場合は、アドレスの **16** 進文字列表記を返します。

---

## 名前

**function::usymline** – アドレスの行番号を返します。

## 概要

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

## 引数

### ***addr***

変換するアドレス。

## 説明

既知の場合、指定のアドレスの (おおよその) 行番号を返します。行番号が不明な場合は、アドレスの 16 進文字列表記を返します。

---

## 名前

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

## 概要

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

## 引数

### ***addr***

変換するアドレス。

## 説明

既知の場合、指定のアドレスに関連する (関数) シンボル名を返します。不明な場合は、アドレスの 16 進文字列表記を返します。

## 第4章 タイムスタンプ関数

各タイムスタンプ関数は、その関数が実行されることを示す値を返します。これらの戻り値は、イベント発生時やイベントの順序を示したり、2つのタイムスタンプ間で経過した時間を算出したりするために使用されます。

### 名前

function::HZ – カーネル HZ

### 概要

```
HZ:long()
```

### 引数

なし

### 説明

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

---

### 名前

function::cpu\_clock\_ms – 指定の CPU クロックに関する時間 (ミリ秒)。

### 概要

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

### 引数

**cpu**

読み取るプロセッサのクロック

### 説明

この関数は、指定の CPU のクロックに関する時間 (ミリ秒単位) を返します。これは、常に同じ CPU での単調な比較になりますが、CPU 間で違いがある場合があります (**jiffy** 内)。

---

### 名前

function::cpu\_clock\_ns – 指定の CPU クロックに関する時間 (ナノ秒)。

### 概要

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

### 引数

**cpu**

読み取るプロセッサのクロック

## 説明

この関数は、指定の **CPU** のクロックに関する時間 (ナノ秒単位) を返します。これは、常に同じ **CPU** での単調な比較になりますが、**CPU** 間で違いがある場合があります (**jiffy** 内)。

## 名前

**function::cpu\_clock\_s** – 指定の **CPU** クロックに関する時間 (秒)

## 概要

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

## 引数

### **cpu**

読み取るプロセッサのクロック

## 説明

この関数は、指定の **CPU** のクロックに関する時間 (秒単位) を返します。これは、常に同じ **CPU** での単調な比較になりますが、**CPU** 間で違いがある場合があります (**jiffy** 内)。

## 名前

**function::cpu\_clock\_us** – 指定の **CPU** クロックに関する時間 (マイクロ秒)

## 概要

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

## 引数

### **cpu**

読み取るプロセッサのクロック

## 説明

この関数は、指定の **CPU** のクロックに関する時間 (マイクロ秒単位) を返します。これは、常に同じ **CPU** での単調な比較になりますが、**CPU** 間で違いがある場合があります (**jiffy** 内)。

## 名前

**function::delete\_stopwatch** – 既存のストップウォッチを削除します。

## 概要

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

## 引数

### *name*

ストップウォッチ名

## 説明

ストップウォッチ **name** を削除します。

---

## 名前

**function::get\_cycles** – プロセッササイクル数

## 概要

```
get_cycles:long()
```

## 引数

なし

## 説明

この関数は、プロセッササイクル数の値を返します。この値がない場合はゼロを返します。サイクルカウンタはフリーランカウンタで、各プロセッサで非同期になります。そのため、異なるプロセッサの **get\_cycles** 関数の結果を比較してもイベントの順序は判断できません。

---

## 名前

**function::gettimeofday\_ms** – UNIX エポックからの経過時間 (ミリ秒)

## 概要

```
gettimeofday_ms:long()
```

## 引数

なし

## 説明

この関数は、UNIX エポックからの経過時間をミリ秒数で返します。

---

## 名前

**function::gettimeofday\_ns** – UNIX エポックからの経過時間 (ナノ秒)

## 概要

```
gettimeofday_ns:long()
```

## 引数

なし

## 説明

この関数は、UNIX エポックからの経過時間をナノ秒数で返します。

---

## 名前

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

## 概要

```
gettimeofday_s:long()
```

## 引数

なし

## 説明

この関数は、UNIX エポックからの経過時間を秒数で返します。

---

## 名前

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

## 概要

```
gettimeofday_us:long()
```

## 引数

なし

## 説明

この関数は、UNIX エポックからの経過時間をマイクロ秒数で返します。

---

## 名前

function::jiffies – カーネルの jiffies 数

## 概要

```
jiffies:long()
```

## 引数

なし

## 説明

この関数は、カーネルの **jiffies** 変数の値を返します。この値はタイマーの中断によって定期的に増分され、**32** ビットまたは **64** ビットの境界をラッピングできます。**HZ** を参照してください。

---

## 名前

**function::local\_clock\_ms** – ローカルの CPU クロックに関する時間 (ミリ秒)。

## 概要

```
local_clock_ms:long()
```

## 引数

なし

## 説明

この関数は、ローカルの **CPU** のクロックに関する時間 (ミリ秒単位) を返します。これは、常に同じ **CPU** での単調な比較になりますが、**CPU** 間で違いがある場合があります (**jiffy** 内)。

---

## 名前

**function::local\_clock\_ns** – ローカルの CPU クロックに関する時間 (ナノ秒)。

## 概要

```
local_clock_ns:long()
```

## 引数

なし

## 説明

この関数は、ローカルの **CPU** のクロックに関する時間 (ナノ秒単位) を返します。これは、常に同じ **CPU** での単調な比較になりますが、**CPU** 間で違いがある場合があります (**jiffy** 内)。

---

## 名前

**function::local\_clock\_s** – ローカルの CPU クロックに関する時間 (秒)。

## 概要

```
local_clock_s:long()
```

## 引数

なし



## 説明

この関数は、ローカルの CPU のクロックに関する時間 (秒単位) を返します。これは、常に同じ CPU での単調な比較になりますが、CPU 間で違いがある場合があります (jiffy 内)。

---

## 名前

`function::local_clock_us` – ローカルの CPU クロックに関する時間 (マイクロ秒)。

## 概要

```
local_clock_us:long()
```

## 引数

なし

## 説明

この関数は、ローカルの CPU のクロックに関する時間 (マイクロ秒単位) を返します。これは、常に同じ CPU での単調な比較になりますが、CPU 間で違いがある場合があります (jiffy 内)。

---

## 名前

`function::read_stopwatch_ms` – ストップウォッチの時間 (ミリ秒単位) を読み取ります。

## 概要

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

## 引数

***name***

ストップウォッチ名

## 説明

ストップウォッチ ***name*** の時間 (ミリ秒単位) を返します。ストップウォッチ ***name*** を作成します (存在しない場合)。

---

## 名前

`function::read_stopwatch_ns` – ストップウォッチの時間 (ナノ秒単位) を読み取ります。

## 概要

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

## 引数

***name***

***name***

ストップウォッチ名

## 説明

ストップウォッチ ***name*** の時間 (ナノ秒単位) を返します。ストップウォッチ ***name*** を作成します (存在しない場合)。

---

## 名前

**function::read\_stopwatch\_s** – ストップウォッチの時間 (秒単位) を読み取ります。

## 概要

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

## 引数

***name***

ストップウォッチ名

## 説明

ストップウォッチ ***name*** の時間 (秒単位) を返します。ストップウォッチ ***name*** を作成します (存在しない場合)。

---

## 名前

**function::read\_stopwatch\_us** – ストップウォッチの時間 (マイクロ秒単位) を読み取ります。

## 概要

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

## 引数

***name***

ストップウォッチ名

## 説明

ストップウォッチ ***name*** の時間 (マイクロ秒単位) を返します。ストップウォッチ ***name*** を作成します (存在しない場合)。

---

## 名前

**function::start\_stopwatch** – ストップウォッチを開始します。

## 概要

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

## 引数

### *name*

ストップウォッチ名

## 説明

ストップウォッチ **name** を開始します。ストップウォッチ **name** を作成します (存在しない場合)。

---

## 名前

`function::stop_stopwatch` – ストップウォッチを停止します。

## 概要

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

## 引数

### *name*

ストップウォッチ名

## 説明

ストップウォッチ **name** を停止します。ストップウォッチ **name** を作成します (存在しない場合)。

## 第5章 時間ユーティリティー関数

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

### 名前

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

### 概要

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

### 引数

#### *epochsecs*

`gettimeofday_s` によって返される、エポックからの経過時間 (秒数)。

### 説明

`gettimeofday_s` によって返されるエポックからの経過時間 (秒数) の引数を取ります。フォームの文字列を返します。

「Wed Jun 30 21:49:08 1993 」

文字列は常に 24 文字になります。時間が大幅に過去のものである場合 (エポックからの経過時間 (秒単位) の 32 ビットオフセットで表すことができる値よりも前の場合) は、エラーが発生します (`try/catch` で回避できます)。時間が大幅に未来のものである場合も、エラーが発生します。

エポック (ゼロ) は以下になります。

「Thu Jan 1 00:00:00 1970 」

`ctime` によって提供される最も古い日付は、`epochsecs 2147483648` に対応する 「Fri Dec 13 20:45:52 1901」 です。`ctime` によって提供される最も新しい日付は、`epochsecs 2147483647` に対応する 「Tue Jan 19 03:14:07 2038」 です。

曜日の省略形は、'Sun'、'Mon'、'Tue'、'Wed'、'Thu'、'Fri'、および 'Sat' です。月の省略形は 'Jan'、'Feb'、'Mar'、'Apr'、'May'、'Jun'、'Jul'、'Aug'、'Sep'、'Oct'、'Nov'、および 'Dec' です。

実際の C ライブラリーの `ctime` 関数は文字列の最後に改行文字 (`\n`) を挿入しますが、この関数は挿入しません。また、カーネルにはタイムゾーンの概念がないため、時間は常に GMT で返されます。

### 名前

**function::tz\_ctime** – エポックからの経過時間 (秒単位) を、人が判読できる日付/時間の文字列にローカルのタイムゾーンで変換します。

### 概要

```
tz_ctime(epochsecs:)
```

## 引数

### *epochsecs*

`gettimeofday_s`によって返される、エポックからの経過時間 (秒数)。

## 説明

`gettimeofday_s`によって返されるエポックからの経過時間 (秒数) の引数を取ります。`ctime`と同じ形式の文字列を返しますが、ローカルタイムゾーンのエポック時間をオフセットし、ローカルタイムゾーンの名前を加えます。文字列の長さは異なることがあります。タイムゾーン情報はスクリプトの起動時にのみ `staprun` により渡されます。

---

## 名前

`function::tz_gmtoff` – ローカルタイムゾーンオフセットを返します。

## 概要

```
tz_gmtoff()
```

## 引数

なし

## 説明

スクリプトの起動時にのみ `staprun` により渡されるローカルタイムゾーンオフセット (UTC から西に数えた秒単位の時間) を返します。

---

## 名前

`function::tz_name` – ローカルタイムゾーン名を返します。

## 概要

```
tz_name()
```

## 引数

なし

## 説明

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

## 第6章 シェルコマンド関数

シェルコマンドをキューに格納するユーティリティー関数。

### 名前

**function::system** – コマンドをシステムに発行します。

### 概要

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

### 引数

#### *cmd*

システムに発行するコマンド

### 説明

この関数はシステムでコマンドを実行します。コマンドは、現在のプローブの完了後にバックグラウンドで起動されます。コマンドは、**stap** または **staprun** コマンドを実行しているユーザーと同じ **UID** で実行されます。

## 第7章 メモリータップセット

この種類のプローブポイントは、メモリー関連のイベントをプローブしたり、現在のプロセスのメモリー使用量を問い合わせたりします。含まれるプローブポイントは次のとおりです。

### 名前

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

### 概要

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

### 引数

#### **addr**

障害が発生しているメモリーアクセスのアドレス

### 説明

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

---

### 名前

**function::bytes\_to\_string** – 指定バイトの人が判読できる文字列。

### 概要

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

### 引数

#### **bytes**

変換するバイト数。

### 説明

バイト数 (1024 バイトが上限)、最後に「K」が付いたキロバイト数 (1024K 未満の場合)、最後に「M」が付いたメガバイト数 (1024M 未満の場合)、または最後に「G」が付いたギガバイト数を表す文字列を返します。K、M、または G が付き、数字が100 未満である場合、「.」と残りが含まれます。返される文字列は 5 文字です (9999G バイトを超える場合や負の値である場合以外は最初に空白文字が挿入されます)。

---

### 名前

**function::mem\_page\_size** – このアーキテクチャーのページのバイト数。

### 概要

■

```
mem_page_size:long()
```

## 引数

なし

---

## 名前

**function::pages\_to\_string** – ページを人が判読できる文字列に変換します。

## 概要

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

## 引数

### *pages*

変換するページ数。

## 説明

ページと **page\_size** を掛けてバイト数を算出し、**bytes\_to\_string** の結果を返します。

---

## 名前

**function::proc\_mem\_data** – ページ単位のプログラムデータサイズ (データ + スタック)。

## 概要

```
proc_mem_data:long()
```

## 引数

なし

## 説明

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

---

## 名前

**function::proc\_mem\_data\_pid** – ページ単位のプログラムデータサイズ (データ + スタック)。

## 概要

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

## 引数

なし



***pid***

確認するプロセスの PID。

**説明**

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

---

**名前**

**function::proc\_mem\_rss** – ページ単位のプログラムレジデント設定サイズ。

**概要**

```
proc_mem_rss:long()
```

**引数**

なし

**説明**

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

---

**名前**

**function::proc\_mem\_rss\_pid** – ページ単位のプログラムレジデント設定サイズ。

**概要**

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

**引数*****pid***

確認するプロセスの PID。

**説明**

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

---

**名前**

**function::proc\_mem\_shr** – プログラム共有ページ (共有マッピングより)。

**概要**

```
proc_mem_shr:long()
```

-

## 引数

なし

## 説明

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

---

## 名前

**function::proc\_mem\_shr\_pid** – プログラム共有ページ (共有マッピングより)。

## 概要

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

## 引数

### *pid*

確認するプロセスの PID。

## 説明

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

---

## 名前

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

## 概要

```
proc_mem_size:long()
```

## 引数

なし

## 説明

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

---

## 名前

**function::proc\_mem\_size\_pid** – ページ単位のプログラム仮想メモリーサイズの合計。

## 概要

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

## 引数

### *pid*

確認するプロセスの PID。

## 説明

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

---

## 名前

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

## 概要

```
proc_mem_string:string()
```

## 引数

なし

## 説明

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

---

## 名前

**function::proc\_mem\_string\_pid** – 人が判読できる文字列で示されたプロセスメモリー使用量。

## 概要

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

## 引数

### *pid*

確認するプロセスの PID。

## 説明

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

---

## 名前

**function::proc\_mem\_txt** – ページ単位のプログラムテキスト (コード) サイズ。

## 概要

```
proc_mem_txt:long()
```

## 引数

なし

## 説明

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

---

## 名前

**function::proc\_mem\_txt\_pid** – ページ単位のプログラムテキスト (コード) サイズ。

## 概要

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

## 引数

### *pid*

確認するプロセスの PID。

## 説明

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

---

## 名前

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

## 概要

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

## 引数

### *value*

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

### *test*

テストするフォールトタイプ (`VM_FAULT_OOM` または同等)。

---

## 名前

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

## 概要

vm.brk

## 値

### **name**

プローブポイントの名前。

### **address**

要求されたアドレス。

### **length**

メモリーセグメントの長さ。

## コンテキスト

brk を呼び出すプロセス。

---

## 名前

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

## 概要

vm.kfree

## 値

### **name**

プローブポイントの名前。

### **ptr**

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

### **caller\_function**

呼び出し元関数の名前。

### **call\_site**

この kmemory 関数を呼び出す関数のアドレス。

---

## 名前

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

## 概要

vm.kmalloc

## 値

### ***gfp\_flags***

割り当てる kmemory のタイプ。

### ***bytes\_req***

要求されたバイト。

### ***name***

プローブポイントの名前。

### ***ptr***

割り当てられた kmemory へのポインター。

### ***bytes\_alloc***

割り当てられたバイト。

### ***caller\_function***

呼び出し元関数の名前。

### ***gfp\_flag\_name***

割り当てる kmemory のタイプ (文字列形式)。

### ***call\_site***

kmemory 関数のアドレス。

---

## 名前

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

## 概要

vm.kmalloc\_node

## 値

### ***caller\_function***

呼び出し元関数の名前。

***gfp\_flag\_name***

割り当てる kmemory のタイプ (文字列形式)。

***call\_site***

この kmemory 関数を呼び出す関数のアドレス。

***gfp\_flags***

割り当てる kmemory のタイプ。

***bytes\_req***

要求されたバイト。

***name***

プローブポイントの名前。

***ptr***

割り当てられた kmemory へのポインター。

***bytes\_alloc***

割り当てられたバイト。

---

## 名前

probe::vm.kmem\_cache\_alloc – kmem\_cache\_alloc が要求されたときに実行されます。

## 概要

vm.kmem\_cache\_alloc

## 値

***bytes\_alloc***

割り当てられたバイト。

***ptr***

割り当てられた kmemory へのポインター。

***name***

プローブポイントの名前。

***bytes\_req***

要求されたバイト。

***gfp\_flags***

割り当てる kmemory のタイプ。

***caller\_function***

呼び出し元関数の名前。

***gfp\_flag\_name***

割り当てる kmemory のタイプ (文字列形式)。

***call\_site***

この kmemory 関数を呼び出す関数のアドレス。

---

## 名前

probe::vm.kmem\_cache\_alloc\_node – kmem\_cache\_alloc\_node が要求されたときに実行されます。

## 概要

```
vm.kmem_cache_alloc_node
```

## 値

***gfp\_flags***

割り当てる kmemory のタイプ。

***name***

プローブポイントの名前。

***bytes\_req***

要求されたバイト。

***ptr***

割り当てられた kmemory へのポインター。

***bytes\_alloc***

割り当てられたバイト。

***caller\_function***

呼び出し元関数の名前。

***call\_site***

この kmemory 関数を呼び出す関数のアドレス。

***gfp\_flag\_name***

割り当てる kmemory のタイプ (文字列形式)。

---



## 名前

probe::vm.kmem\_cache\_free – kmem\_cache\_free が要求されたときに実行されます。

## 概要

```
vm.kmem_cache_free
```

## 値

### **caller\_function**

呼び出し元関数の名前。

### **call\_site**

この kmemory 関数を呼び出す関数のアドレス。

### **ptr**

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

### **name**

プローブポイントの名前。

---

## 名前

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

## 概要

```
vm.mmap
```

## 値

### **name**

プローブポイントの名前。

### **length**

メモリーセグメントの長さ。

### **address**

要求されたアドレス。

## コンテキスト

mmap を呼び出すプロセス。

---

## 名前

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

## 概要

`vm.munmap`

## 値

### *length*

メモリーセグメントの長さ。

### *address*

要求されたアドレス。

### *name*

プローブポイントの名前。

## コンテキスト

`munmap` を呼び出すプロセス。

---

## 名前

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

## 概要

`vm.oom_kill`

## 値

### *name*

プローブポイントの名前。

### *task*

Kill されるタスク。

## コンテキスト

余分なメモリーを消費しようとし、**OOM** を引き起こしたプロセス。

---

## 名前

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

## 概要

`vm.pagefault`

## 値

### **address**

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

### **write\_access**

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

### **name**

プローブポイントの名前。

## コンテキスト

障害を引き起こしたプロセス。

---

## 名前

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

## 概要

`vm.pagefault.return`

## 値

### **name**

プローブポイントの名前。

### **fault\_type**

メモリー不足による障害は 0 (VM\_FAULT\_OOM)、小さな障害は 2 (VM\_FAULT\_MINOR)、大きな障害は 3 (VM\_FAULT\_MAJOR) を返します。障害がいずれにも該当しない場合は 1 (VM\_FAULT\_SIGBUS) を返します。

---

## 名前

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

## 概要

`vm.write_shared`

## 値

### **address**

共有書き込みのアドレス。

***name***

プローブポイントの名前。

## コンテキスト

コンテキストは、書き込みを試行するプロセスです。

## 説明

プロセスが共有ページへの書き込みを試行すると実行されます。コピーが必要な場合、この後に `vm.write_shared_copy` が実行されます。

---

## 名前

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

## 概要

`vm.write_shared_copy`

## 値

***zero***

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

***name***

プローブポイントの名前。

***address***

共有書き込みのアドレス。

## コンテキスト

書き込みを試行するプロセス。

## 説明

共有ページへの書き込みでページのコピーが必要な場合に実行されます。常に `vm.write_shared` の後に実行されます。

## 第8章 タスク時間タップセット

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

### 名前

**function::cputime\_to\_msecs** – 指定の CPU 時間をミリ秒に変換します。

### 概要

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

### 引数

#### ***cputime***

ミリ秒に変換する時間。

---

### 名前

**function::cputime\_to\_string** – 人が判読できる文字列で示された指定の CPU 時間。

### 概要

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

### 引数

#### ***cputime***

変換する時間。

### 説明

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

---

### 名前

**function::cputime\_to\_usecs** – 指定の CPU 時間をマイクロ秒に変換します。

### 概要

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

### 引数

#### ***cputime***

マイクロ秒に変換する時間。

---

## 名前

**function::msecs\_to\_string** – 人が判読できる文字列で示された指定のミリ秒。

## 概要

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

## 引数

### *msecs*

変換するミリ秒数。

## 説明

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

---

## 名前

**function::nsecs\_to\_string** – 人が判読できる文字列で示された指定のナノ秒。

## 概要

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

## 引数

### *nsecs*

変換するナノ秒数。

## 説明

ナノ秒数を人が判読できる文字列で返します。この文字列の形式は「XmY.ZZZZZZs」で、Xは分数、Yは秒数、ZZZZZZZZはナノ秒数を示します。

---

## 名前

**function::task\_start\_time** – 指定タスクの開始時間。

## 概要

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

## 引数

### *tid*

指定タスクのスレッド ID。

## 説明

指定タスクの開始時間を起動時間以降のナノ秒で返し、タスクが存在しない場合は 0 を返します。

---

## 名前

`function::task_stime` – 現在のタスクのシステム時間。

## 概要

```
task_stime:long()
```

## 引数

なし

## 説明

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

---

## 名前

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

## 概要

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

## 引数

***tid***

指定タスクのスレッド ID。

## 説明

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

---

## 名前

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

## 概要

```
task_time_string:string()
```

## 引数

なし

## 説明

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

---

## 名前

**function::task\_time\_string\_tid** – 人が判読できる文字列で示されたタスク時間の使用量。

## 概要

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

## 引数

***tid***

指定タスクのスレッド ID。

## 説明

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

---

## 名前

**function::task\_utime** – 現在のタスクのユーザー時間。

## 概要

```
task_utime:long()
```

## 引数

なし

## 説明

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

---

## 名前

**function::task\_utime\_tid** – 指定タスクのユーザー時間。

## 概要

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



## 引数

### *tid*

指定タスクのスレッド ID。

## 説明

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

---

## 名前

`function::usecs_to_string` – 人が判読できる文字列で示された指定のナノ秒。

## 概要

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

## 引数

### *usecs*

変換するミリ秒数。

## 説明

マイクロ秒数を人が判読できる文字列で返します。この文字列の形式は「`XmY.ZZZZZZs`」で、`X` は分数、`Y` は秒数、`ZZZZZZ` はマイクロ秒数を示します。

## 第9章 スケジューラータップセット

この種類のプローブポイントは、タスクスケジューラーの活動をプローブするために使用されます。以下のプローブポイントが含まれます。

### 名前

`probe::scheduler.balance` – より多くのワークを見つけようとする CPU。

### 概要

```
scheduler.balance
```

### 値

#### *name*

プローブポイントの名前。

### コンテキスト

より多くのワークを探している CPU。

---

### 名前

`probe::scheduler.cpu_off` – プロセスが CPU で実行を停止します。

### 概要

```
scheduler.cpu_off
```

### 値

#### *task\_prev*

CPU を出るプロセス (現在プロセスと同じ)。

#### *idle*

`current` がアイドル状態のプロセスであるかどうかを示すブール値。

#### *name*

プローブポイントの名前。

#### *task\_next*

`current` を置き換えるプロセス。

### コンテキスト

CPU を脱退するプロセス。

---

## 名前

probe::scheduler.cpu\_on – プロセスが CPU で実行を開始します。

## 概要

`scheduler.cpu_on`

## 値

### **idle**

- `current` がアイドル状態のプロセスであるかどうかを示すブール値。

### **task\_prev**

この CPU で以前に実行されていたプロセス。

### **name**

プローブポイントの名前。

## コンテキスト

再開するプロセス。

---

## 名前

probe::scheduler.ctxswitch – コンテキストスイッチが行われています。

## 概要

`scheduler.ctxswitch`

## 値

### **prev\_tid**

スイッチアウトされるプロセスの TID。

### **name**

プローブポイントの名前。

### **next\_tid**

スイッチインされるプロセスの TID。

### **prev\_pid**

スイッチアウトされるプロセスの PID。

### **prevtsk\_state**

スイッチアウトされるプロセスの状態。

***next\_pid***

スイッチインされるプロセスの PID。

***nexttsk\_state***

スイッチインされるプロセスの状態。

***prev\_priority***

スイッチアウトされるプロセスの優先度。

***next\_priority***

スイッチインされるプロセスの優先度。

***prev\_task\_name***

スイッチアウトされるプロセスの名前。

***next\_task\_name***

スイッチインされるプロセスの名前。

---

## 名前

probe::scheduler.kthread\_stop – kthread\_create により作成されたスレッドが停止されます。

## 概要

```
scheduler.kthread_stop
```

## 値

***thread\_pid***

停止するスレッドの PID。

***thread\_priority***

スレッドの優先度。

---

## 名前

probe::scheduler.kthread\_stop.return – kthread が停止され、戻り値を取得します。

## 概要

```
scheduler.kthread_stop.return
```

## 値

***return value***

---

`probe::scheduler`

スレッドの停止後の戻り値

***name***

プローブポイントの名前。

---

## 名前

`probe::scheduler.migrate` – CPU を移行するタスク

## 概要

`scheduler.migrate`

## 値

***priority***

移行するタスクの優先度

***cpu\_to***

宛先 CPU

***cpu\_from***

元の CPU

***task***

移行されるプロセス

***name***

プローブポイントの名前。

***pid***

移行されるタスクの PID

---

## 名前

`probe::scheduler.process_exit` – 終了するプロセス

## 概要

`scheduler.process_exit`

## 値

***name***

プローブポイントの名前。

***pid***

終了するプロセスの PID

***priority***

終了するプロセスの優先度

---

## 名前

probe::scheduler.process\_fork – フォークされるプロセス

## 概要

```
 scheduler.process_fork
```

## 値

***name***

プローブポイントの名前。

***parent\_pid***

親プロセスの PID

***child\_pid***

子プロセスの PID

---

## 名前

probe::scheduler.process\_free – プロセスに対してデータ構造を解放するスケジューラー

## 概要

```
 scheduler.process_free
```

## 値

***name***

プローブポイントの名前。

***pid***

解放されるプロセスの PID

***priority***

解放されるプロセスの優先度

## 名前

`probe::scheduler.process_wait` – プロセスで待機を開始するスケジューラ

## 概要

```
scheduler.process_wait
```

## 値

### *name*

プローブポイントの名前。

### *pid*

プロセススケジューラが待機している PID

## 名前

`probe::scheduler.signal_send` – シグナルの送信

## 概要

```
scheduler.signal_send
```

## 値

### *pid*

シグナルを送信するプロセスの PID

### *name*

プローブポイントの名前。

### *signal\_number*

シグナル番号

## 名前

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

## 概要

■

`scheduler.tick`

## 値

### ***idle***

`current` がアイドル状態のプロセスであるかどうかを示すブール値。

### ***name***

プローブポイントの名前。

## コンテキスト

アカウンティングが更新されるプロセス。

---

## 名前

`probe::scheduler.wait_task` – スケジュールを解除する (無効になる) タスクを待機しています。

## 概要

`scheduler.wait_task`

## 値

### ***task\_pid***

スケジューラーが待機しているタスクの PID

### ***name***

プローブポイントの名前。

### ***task\_priority***

タスクの優先度

---

## 名前

`probe::scheduler.wakeup` – タスクがウェイクアップします。

## 概要

`scheduler.wakeup`

## 値

### ***task\_tid***

ウェイクアップするタスクの TID

---



***task\_priority***

ウェイクアップするタスクの優先度

***task\_cpu***

ウェイクアップするタスクの CPU

***task\_pid***

ウェイクアップするタスクの PID

***name***

プローブポイントの名前。

***task\_state***

ウェイクアップするタスクの状態

---

## 名前

probe::scheduler.wakeup\_new – 新しく作成されたタスクが初めてウェイクアップします。

## 概要

`scheduler.wakeup_new`

## 値

***name***

プローブポイントの名前。

***task\_state***

ウェイクアップしたタスクの状態

***task\_pid***

ウェイクアップした新しいタスクの PID

***task\_tid***

ウェイクアップした新しいタスクの TID

***task\_priority***

新しいタスクの優先度

***task\_cpu***

ウェイクアップしたタスクの CPU

## 第10章 IO スケジューラーおよびブロック IO タップセット

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

### 名前

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

### 概要

`ioblock.end`

### 値

#### ***name***

プローブポイントの名前。

#### ***sector***

bio 全体の開始セクター

#### ***hw\_segments***

物理および DMA 再マッピングハードウェアコアレスシングが実行された後のセグメントの数

#### ***phys\_segments***

物理アドレスコアレスシングが実行された後のこの bio のセグメント数。

#### ***flags***

以下を参照。BIO\_UPTODATE 0 I/O 完了後 ok、BIO\_RW\_BLOCK 1 RW\_AHEAD セット、読み取り/書き込みのブロック、BIO\_EOF 2 out-out-bounds エラー、BIO\_SEG\_VALID 3 nr\_hw\_seg が有効、BIO\_CLONED 4 データを所有しない、BIO\_BOUNCED 5 bio はバウンス bio、BIO\_USER\_MAPPED 6 ユーザーページを含む、BIO\_EOPNOTSUPP 7 サポートされない。

#### ***devname***

ブロックデバイス名

#### ***bytes\_done***

転送済みバイト数

#### ***error***

0 - 成功

#### ***size***

合計サイズ (バイト単位)

#### ***idx***

bio ベクターアレイへのオフセット

***vcnt***

この I/O 要求を構成するアレイ要素 (ページ、オフセット、長さ) の数を表す **bio** ベクター数

***ino***

マップされたファイルの **i-node** 番号

***rw***

読み取り/書き込み要求のバイナリトレース

## コンテキスト

プロセスは転送が行われたことを示します。

## 名前

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

## 概要

```
ioblock.request
```

## 値

***sector***

**bio** 全体の開始セクター

***name***

プローブポイントの名前。

***devname***

ブロックデバイス名

***phys\_segments***

物理アドレスコアレスシングが実行された後のこの **bio** のセグメント数

***flags***

以下を参照。 **BIO\_UPTODATE** 0 I/O 完了後 ok、 **BIO\_RW\_BLOCK** 1 **RW\_AHEAD** セット、読み取り/書き込みのブロック、 **BIO\_EOF** 2 out-out-bounds エラー、 **BIO\_SEG\_VALID** 3 **nr\_hw\_seg** が有効、 **BIO\_CLONED** 4 データを所有しない、 **BIO\_BOUNCED** 5 **bio** はバウンス **bio**、 **BIO\_USER\_MAPPED** 6 ユーザーページを含む、 **BIO\_EOPNOTSUPP** 7 サポートされない。

***hw\_segments***

物理および DMA 再マッピングハードウェアコアレスシングが実行された後のセグメントの数

***bdev\_contains***

パーティションを含むデバイスオブジェクトを参照します (**bio** 構造がパーティションを表す場合)。

***vcnt***

この I/O 要求を構成するアレイ要素 (ページ、オフセット、長さ) の数を表す **bio** ベクター数。

***idx***

**bio** ベクターアレイへのオフセット

***bdev***

ターゲットブロックデバイス

***p\_start\_sect***

デバイスのパーティション構造の開始セクターを参照します。

***size***

合計サイズ (バイト単位)

***ino***

マップされたファイルの i-node 番号

***rw***

読み取り/書き込み要求のバイナリートレース

## コンテキスト

ブロック I/O 要求を行うプロセス

---

## 名前

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

## 概要

**ioblock\_trace.bounce**

## 値

***q***

この **bio** が格納された要求キュー。

***size***

合計サイズ (バイト単位)

***vcnt***

この I/O 要求を構成するアレイ要素 (ページ、オフセット、長さ) の数を表す **bio** ベクター数

***idx***

**bio** ベクターアレイ **phys\_segments** へのオフセット - 物理アドレスコアレスシングが実行された後のこの **bio** のセグメント数。

**bdev**

ターゲットブロックデバイス

**p\_start\_sect**

デバイスのパーティション構造の開始セクターを参照します。

**ino**

マップされたファイルの i-node 番号

**rw**

読み取り/書き込み要求のバイナリートレース

**name**

プローブポイントの名前。

**sector**

**bio** 全体の開始セクター

**bdev\_contains**

パーティションを含むデバイスオブジェクトを参照します (**bio** 構造がパーティションを表す場合)。

**devname**

バッファバウンスが必要なデバイス。

**flags**

以下を参照。BIO\_UPTODATE 0 I/O 完了後 ok、BIO\_RW\_BLOCK 1 RW\_AHEAD セット、読み取り/書き込みのブロック、BIO\_EOF 2 out-out-bounds エラー、BIO\_SEG\_VALID 3 nr\_hw\_seg が有効、BIO\_CLONED 4 データを所有しない、BIO\_BOUNCED 5 bio はバウンス bio、BIO\_USER\_MAPPED 6 ユーザーページを含む、BIO\_EOPNOTSUPP 7 サポートされない。

**bytes\_done**

転送済みバイト数

**コンテキスト**

ブロック I/O 要求を作成するプロセス。

**名前**

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

**概要**

```
ioblock_trace.end
```

## 値

### ***bdev\_contains***

パーティションを含むデバイスオブジェクトを参照します (**bio** 構造がパーティションを表す場合)。

### ***flags***

以下を参照。BIO\_UPTODATE 0 I/O 完了後 ok、BIO\_RW\_BLOCK 1 RW\_AHEAD セット、読み取り/書き込みのブロック、BIO\_EOF 2 out-out-bounds エラー、BIO\_SEG\_VALID 3 nr\_hw\_seg が有効、BIO\_CLONED 4 データを所有しない、BIO\_BOUNCED 5 bio はバウンス bio、BIO\_USER\_MAPPED 6 ユーザーページを含む、BIO\_EOPNOTSUPP 7 サポートされない。

### ***devname***

ブロックデバイス名

### ***bytes\_done***

転送済みバイト数

### ***name***

プローブポイントの名前。

### ***sector***

bio 全体の開始セクター

### ***ino***

マップされたファイルの i-node 番号

### ***rw***

読み取り/書き込み要求のバイナリートレース

### ***size***

合計サイズ (バイト単位)

### ***q***

この bio が格納された要求キュー。

### ***idx***

bio ベクターアレイ **phys\_segments** へのオフセット - 物理アドレスコアレスシングが実行された後のこの bio のセグメント数。

### ***vcnt***

この I/O 要求を構成するアレイ要素 (ページ、オフセット、長さ) の数を表す bio ベクター数

### ***bdev***

ターゲットブロックデバイス

### ***p\_start\_sect***

デバイスのパーティション構造の開始セクターを参照します。

## コンテキスト

プロセスは転送が行われたことを示します。

---

## 名前

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

## 概要

**ioblock\_trace.request**

## 値

### ***q***

この **bio** が格納された要求キュー。

### ***size***

合計サイズ (バイト単位)

### ***idx***

**bio** ベクターアレイ **phys\_segments** へのオフセット - 物理アドレスコアレスシングが実行された後のこの **bio** のセグメント数。

### ***vcnt***

この I/O 要求を構成するアレイ要素 (ページ、オフセット、長さ) の数を表す **bio** ベクター数。

### ***bdev***

ターゲットブロックデバイス

### ***p\_start\_sect***

デバイスのパーティション構造の開始セクターを参照します。

### ***ino***

マップされたファイルの i-node 番号

### ***rw***

読み取り/書き込み要求のバイナリートレース

### ***name***

プローブポイントの名前。

**sector**

bio 全体の開始セクター

**bdev\_contains**

パーティションを含むデバイスオブジェクトを参照します (bio 構造がパーティションを表す場合)。

**devname**

ブロックデバイス名

**flags**

以下を参照。BIO\_UPTODATE 0 I/O 完了後 ok、BIO\_RW\_BLOCK 1 RW\_AHEAD セット、読み取り/書き込みのブロック、BIO\_EOF 2 out-out-bounds エラー、BIO\_SEG\_VALID 3 nr\_hw\_seg が有効、BIO\_CLONED 4 データを所有しない、BIO\_BOUNCED 5 bio はバウンス bio、BIO\_USER\_MAPPED 6 ユーザーページを含む、BIO\_EOPNOTSUPP 7 サポートされない。

**bytes\_done**

転送済みバイト数

**コンテキスト**

ブロック I/O 要求を行うプロセス

---

**名前**

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

**概要**

```
ioscheduler.elv_add_request
```

**値****rq**

要求のアドレス。

**q**

要求キューのポインター。

**elevator\_name**

現在有効になっている I/O エレベーターのタイプ。

**disk\_major**

要求のディスクメジャー番号。

**disk\_minor**

要求のディスクマイナー番号。



***rq\_flags***

要求フラグ。

---

**名前**

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

**概要**

```
ioscheduler.elv_add_request.kp
```

**値*****disk\_major***

要求のディスクメジャー番号。

***disk\_minor***

要求のディスクマイナー番号。

***rq\_flags***

要求フラグ。

***elevator\_name***

現在有効になっている I/O エレベーターのタイプ。

***q***

要求キューへのポインター。

***rq***

要求のアドレス。

***name***

プローブポイントの名前。

---

**名前**

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

**概要**

```
ioscheduler.elv_add_request.tp
```

## 値

### ***q***

要求キューのポインター。

### ***elevator\_name***

現在有効になっている I/O エレベーターのタイプ。

### ***name***

プローブポイントの名称。

### ***rq***

要求のアドレス。

### ***disk\_major***

要求のディスクメジャー番号。

### ***disk\_minor***

要求のディスクマイナー番号。

### ***rq\_flags***

要求フラグ。

---

## 名前

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

## 概要

```
ioscheduler.elv_completed_request
```

## 値

### ***name***

プローブポイントの名称。

### ***rq***

要求のアドレス。

### ***elevator\_name***

現在有効になっている I/O エレベーターのタイプ。

### ***disk\_major***

要求のディスクメジャー番号。

***disk\_minor***

要求のディスクマイナー番号。

***rq\_flags***

要求フラグ。

## 名前

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

## 概要

```
ioscheduler.elv_next_request
```

## 値

***elevator\_name***

現在有効になっている I/O エレベーターのタイプ。

***name***

プローブポイントの名前。

## 名前

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

## 概要

```
ioscheduler.elv_next_request.return
```

## 値

***disk\_major***

要求のディスクメジャー番号。

***disk\_minor***

要求のディスクマイナー番号。

***rq\_flags***

要求フラグ。

***rq***

要求のアドレス。

***name***

プローブポイントの名前。

---

**名前**

probe::ioscheduler\_trace.elv\_abort\_request – 要求が中止されると実行されます。

**概要**

```
ioscheduler_trace.elv_abort_request
```

**値*****disk\_major***

要求のディスクメジャー番号。

***disk\_minor***

要求のディスクマイナー番号。

***rq\_flags***

要求フラグ。

***elevator\_name***

現在有効になっている I/O エレベーターのタイプ。

***rq***

要求のアドレス。

***name***

プローブポイントの名前。

---

**名前**

probe::ioscheduler\_trace.elv\_completed\_request – 要求が完了すると実行されます。

**概要**

```
ioscheduler_trace.elv_completed_request
```

**値*****elevator\_name***

現在有効になっている I/O エレベーターのタイプ。

***rq***

要求のアドレス。

***name***

プローブポイントの名前。

***rq\_flags***

要求フラグ。

***disk\_minor***

要求のディスクマイナー番号。

***disk\_major***

要求のディスクメジャー番号。

## 説明

完了済み。

---

## 名前

probe::ioscheduler\_trace.elv\_issue\_request – 要求が完了すると実行されます。

## 概要

```
ioscheduler_trace.elv_issue_request
```

## 値

***rq\_flags***

要求フラグ。

***disk\_minor***

要求のディスクマイナー番号。

***disk\_major***

要求のディスクメジャー番号。

***elevator\_name***

現在有効になっている I/O エレベーターのタイプ。

***rq***

要求のアドレス。

***name***

プローブポイントの名前。

## 説明

スケジュール済み。

---

## 名前

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

## 概要

`ioscheduler_trace.elv_requeue_request`

## 値

### ***rq***

要求のアドレス。

### ***name***

プローブポイントの名前。

### ***elevator\_name***

現在有効になっている I/O エレベーターのタイプ。

### ***rq\_flags***

要求フラグ。

### ***disk\_minor***

要求のディスクマイナー番号。

### ***disk\_major***

要求のディスクメジャー番号。

## 説明

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

---

## 名前

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

## 概要

`ioscheduler_trace.plug`

## 値

### ***rq\_queue***

要求キュー

***name***

プローブポイントの名前。

**説明**

キュー内の要求はブロックドライバーで処理できません。

---

**名前**

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

**概要**

`ioscheduler_trace.unplug_io`

**値*****name***

プローブポイントの名前。

***rq\_queue***

要求キュー

**説明**

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

---

**名前**

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

**概要**

`ioscheduler_trace.unplug_timer`

**値*****rq\_queue***

要求キュー

***name***

プローブポイントの名前。

**説明**

要求キューの終了時。



## 第11章 SCSI タップセット

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

### 名前

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

### 概要

`scsi.iocompleted`

### 値

#### ***device\_state***

デバイスの現在の状態

#### ***dev\_id***

SCSI デバイス ID

#### ***req\_addr***

現在の構造要求ポインター (数字)

#### ***data\_direction\_str***

データの方向 (文字列)

#### ***device\_state\_str***

デバイスの現在の状態 (文字列)

#### ***lun***

LUN 番号

#### ***goodbytes***

完了済みバイト数

#### ***data\_direction***

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

#### ***channel***

チャンネル番号

#### ***host\_no***

ホスト番号

## 名前

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

## 概要

`scsi.iodispatching`

## 値

### ***device\_state***

デバイスの現在の状態

### ***request\_bufflen***

要求バッファの長さ

### ***request\_buffer***

要求バッファアドレス

### ***dev\_id***

SCSI デバイス ID

### ***data\_direction\_str***

データの方法 (文字列)

### ***req\_addr***

現在の構造要求ポインター (数字)

### ***device\_state\_str***

デバイスの現在の状態 (文字列)

### ***lun***

LUN 番号

### ***data\_direction***

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

### ***channel***

チャンネル番号

### ***host\_no***

ホスト番号

## 名前

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

## 概要

`scsi.iodone`

## 値

### ***device\_state***

デバイスの現在の状態

### ***data\_direction\_str***

データの方法 (文字列)

### ***req\_addr***

現在の構造要求ポインター (数字)

### ***dev\_id***

SCSI デバイス ID

### ***lun***

LUN 番号

### ***scsi\_timer\_pending***

タイマーがこの要求で保留中の場合は 1

### ***device\_state\_str***

デバイスの現在の状態 (文字列)

### ***host\_no***

ホスト番号

### ***channel***

チャネル番号

### ***data\_direction***

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

---

## 名前

`probe::scsi.ioentry` – SCSI 中間層要求の作成

## 概要

`scsi.ioentry`

## 値

### ***req\_addr***

現在の構造要求ポインター (数字)

### ***disk\_major***

ディスクのメジャー番号 (未指定の場合は -1)

### ***device\_state\_str***

デバイスの現在の状態 (文字列)

### ***disk\_minor***

ディスクのマイナー番号 (未指定の場合は -1)

### ***device\_state***

デバイスの現在の状態

---

## 名前

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

## 概要

`scsi.ioexecute`

## 値

### ***host\_no***

ホスト番号

### ***channel***

チャンネル番号

### ***data\_direction***

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

### ***lun***

LUN 番号

### ***retries***

要求を再試行する回数

***device\_state\_str***

デバイスの現在の状態 (文字列)

***data\_direction\_str***

データの方向 (文字列)

***dev\_id***

SCSI デバイス ID

***request\_buffer***

データバッファアドレス

***request\_bufflen***

データバッファの長さ

***device\_state***

デバイスの現在の状態

***timeout***

要求タイムアウト (秒単位)

---

## 名前

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

## 概要

```
scsi.set_state
```

## 値

***state***

デバイスの新しい状態

***old\_state***

デバイスの現在の状態

***dev\_id***

SCSI デバイス ID

***state\_str***

デバイスの新しい状態 (文字列)

***old\_state\_str***

デバイスの現在の状態 (文字列)

***lun***

LUN 番号

***channel***

チャンネル番号

***host\_no***

ホスト番号

## 第12章 TTY タップセット

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

### 名前

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

### 概要

`tty.init`

### 値

#### *name*

ドライバー `.dev_name` の名前

#### *module*

モジュール名

#### *driver\_name*

ドライバー名

### 名前

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

### 概要

`tty.ioctl`

### 値

#### *arg*

`ioctl` 引数

#### *name*

ファイル名

#### *cmd*

`ioctl` コマンド

### 名前

`probe::tty.open` – `tty` が開かれると呼び出されます。

## 概要

`tty.open`

## 値

***inode\_state***

inode 状態

***file\_mode***

ファイルモード

***inode\_number***

inode 番号

***file\_flags***

ファイルフラグ

***file\_name***

ファイル名

***inode\_flags***

inode フラグ

---

## 名前

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

## 概要

`tty.poll`

## 値

***file\_name***

`tty` ファイル名

***wait\_key***

待機キューキー

---

## 名前



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

## 概要

`tty.read`

## 値

***file\_name***

`tty` に関連するファイル名

***driver\_name***

ドライバー名

***nr***

読み込まれる文字数

***buffer***

文字を受信するバッファ

---

## 名前

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

## 概要

`tty.receive`

## 値

***driver\_name***

ドライバー名

***count***

受信する文字数

***index***

`tty` インデックス

***cp***

受信したバッファ

***id***

`tty id`

***name***

モジュールファイルの名前

***fp***

フラグバッファ

---

## 名前

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

## 概要

**|** tty.register

## 値

***name***

ドライバー .dev\_name の名前

***module***

モジュール名

***index***

要求される tty インデックスです。

***driver\_name***

ドライバー名

---

## 名前

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

## 概要

**|** tty.release

## 値

***inode\_flags***

inode フラグ

***file\_flags***

ファイルフラグ

***file\_name***

ファイル名

***inode\_state***

inode 状態

***inode\_number***

inode 番号

***file\_mode***

ファイルモード

---

## 名前

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

## 概要

■ tty.resize

## 値

***new\_row***

新規の行の値

***old\_row***

古い行の値

***name***

tty 名

***new\_col***

新規の列の値

***old\_xpixel***

古い xpixel

***old\_col***

古い列の値

***new\_xpixel***

新規の xpixel 値

***old\_ypixel***

古い ypixel

***new\_ypixel***

新規の `ypixel` 値

---

**名前**

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

**概要**

```
| tty.unregister
```

**値*****name***

ドライバー `.dev_name` の名前

***module***

モジュール名

***index***

要求される `tty` インデックスです。

***driver\_name***

ドライバー名

---

**名前**

`probe::tty.write` – `tty` 行に書き込みます。

**概要**

```
| tty.write
```

**値*****nr***

文字数

***buffer***

書き込まれるバッファ

***file\_name***

`tty` に関連するファイル名

*driver\_name*

ドライバー名

## 第13章 割り込み要求 (IRQ) タップセット

この種類のプローブポイントは、割り込み要求 (IRQ) の活動をプローブするために使用されます。以下のプローブポイントが含まれます。

### 名前

probe::irq\_handler.entry – 割り込みハンドラーの実行

### 概要

```
irq_handler.entry
```

### 値

#### **next\_irqaction**

共有割り込みの次の irqaction のポインター

#### **thread\_fn**

スレッドされた割り込みの割り込みハンドラー関数

#### **thread**

スレッドされた割り込みのスレッドポインター

#### **thread\_flags**

スレッドに関するフラグ

#### **irq**

IRQ 番号

#### **flags\_str**

IRQ フラグのシンボリック文字列表現

#### **dev\_name**

デバイスの名前

#### **action**

この割り込み番号に対する struct irqaction\*

#### **dir**

proc/irq/NN/name エントリーのポインター

#### **flags**

IRQ ハンドラーのフラグ

#### **dev\_id**

デバイスを識別するクッキー。

**handler**

割り込みハンドラー関数

---

**名前**

probe::irq\_handler.exit – 割り込みハンドラーの実行が完了しました。

**概要**

`irq_handler.exit`

**値****flags\_str**

IRQ フラグのシンボリック文字列表現

**dev\_name**

デバイスの名前

**ret**

ハンドラーの戻り値

**action**

struct irqaction\*

**thread\_fn**

スレッドされた割り込みの割り込みハンドラー関数

**next\_irqaction**

共有割り込みの次の irqaction のポインター

**thread**

スレッドされた割り込みのスレッドポインター

**thread\_flags**

スレッドに関するフラグ

**irq**

割り込み番号

**handler**

実行された割り込みハンドラー関数

**flags**

IRQ ハンドラーのフラグ

***dir***

proc/irq/NN/name エントリーのポインター

***dev\_id***

デバイスを識別するクッキー。

---

## 名前

probe::softirq.entry – 保留中の **softirq** ハンドラーの実行が開始されます。

## 概要

**softirq.entry**

## 値

***action***

すぐに実行される **softirq** ハンドラーのポインター

***vec\_nr***

**softirq** ベクター番号

***vec***

**softirq\_action** ベクター

***h***

現在の保留中 **softirq** の **struct softirq\_action\***

---

## 名前

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

## 概要

**softirq.exit**

## 値

***vec\_nr***

**softirq** ベクター番号

***action***

実行を完了したばかりの **softirq** ハンドラーのポインター。



***h***実行された `softirq` の `struct softirq_action*`***vec***`softirq_action` ベクター

## 名前

`probe::workqueue.create` – 新規ワークキューの作成

## 概要

`workqueue.create`

## 値

***wq\_thread***ワークキュースレッドの `task_struct`***cpu***

ワーカースレッドが作成された CPU

## 名前

`probe::workqueue.destroy` – ワークキューの破棄

## 概要

`workqueue.destroy`

## 値

***wq\_thread***ワークキュースレッドの `task_struct`

## 名前

`probe::workqueue.execute` – 遅延されたワークの実行

## 概要

`workqueue.execute`

## 値

### ***wq\_thread***

ワークキュースレッドの `task_struct`

### ***work\_func***

ハンドラー関数のポインター

### ***work***

実行される `work_struct*`

---

## 名前

`probe::workqueue.insert` – ワークキューにワークを格納します。

## 概要

```
workqueue.insert
```

## 値

### ***wq\_thread***

ワークキュースレッドの `task_struct`

### ***work\_func***

ハンドラー関数のポインター

### ***work***

キューに格納される `work_struct*`

## 第14章 ネットワーキングタップセット

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

### 名前

**function::format\_ipaddr** – IP アドレスを表す文字列を返します。

### 概要

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

### 引数

#### **addr**

IP アドレス。

#### **family**

IP アドレスの種類 (AF\_INET または AF\_INET6)。

---

### 名前

**function::htonl** – 32 ビット long をホストからネットワークの順序に変換します。

### 概要

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

### 引数

#### **x**

変換する値

---

### 名前

**function::htonll** – 64 ビット long をホストからネットワークの順序に変換します。

### 概要

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

### 引数

#### **x**

変換する値

## 名前

**function::htons** – 16 ビット **short** をホストからネットワークの順序に変換します。

## 概要

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

## 引数

**x**

変換する値

---

## 名前

**function::ip\_ntop** – IPv4 アドレスを表す文字列を返します。

## 概要

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

## 引数

**addr**

整数として表される IPv4 アドレス。

---

## 名前

**function::ntohl** – 32 ビット **long** をネットワークからホストの順序に変換します。

## 概要

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

## 引数

**x**

変換する値

---

## 名前

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

## 概要

```
ntohl1:long(x:long)
```

## 引数

**x**

変換する値

---

## 名前

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

## 概要

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

## 引数

**x**

変換する値

---

## 名前

**probe::netdev.change\_mac** – **netdev\_name** により **MAC** が変更されたときに呼び出されます。

## 概要

```
netdev.change_mac
```

## 値

**mac\_len**

MAC 長

**old\_mac**

現在の **MAC** アドレス

**dev\_name**

MAC が変更されたデバイス

**new\_mac**

新しい **MAC** アドレス

---

## 名前

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

## 概要

```
netdev.change_mtu
```

## 値

### *old\_mtu*

現在の MTU

### *new\_mtu*

新しい MTU

### *dev\_name*

MTU が変更されたデバイス

---

## 名前

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

## 概要

```
netdev.change_rx_flag
```

## 値

### *flags*

新しいフラグ

### *dev\_name*

変更されるデバイス

---

## 名前

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

## 概要

```
netdev.close
```

## 値

### *dev\_name*

閉じられるデバイス

---

## 名前

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

## 概要

`netdev.get_stats`

## 値

### *dev\_name*

統計を提供するデバイス

---

## 名前

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

## 概要

`netdev.hard_transmit`

## 値

### *true\_size*

送信されるデータのサイズ

### *dev\_name*

送信するようスケジュールされたデバイス

### *protocol*

送信で使用するプロトコル

### *length*

送信バッファの長さ。

---

## 名前

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

## 概要

`netdev.ioctl`

値

***arg***

IOCTL 引数 (通常は `netdev` インターフェース)

***cmd***

IOCTL 要求

---

名前

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

概要

`netdev.open`

値

***dev\_name***

開くデバイス

---

名前

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

概要

`netdev.receive`

値

***length***

受信バッファの長さ。

***protocol***

受信されるパケットのプロトコル。

***dev\_name***

デバイスの名前 (例: `eth0`、`ath1`)。

---



## 名前

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

## 概要

`netdev.register`

## 値

### ***dev\_name***

登録されるデバイス

---

## 名前

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

## 概要

`netdev.rx`

## 値

### ***dev\_name***

パケットを受信したデバイス。

### ***protocol***

パケットプロトコル

---

## 名前

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

## 概要

`netdev.set_promiscuity`

## 値

### ***dev\_name***

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

### ***enable***

デバイスでプロミスカスモードが開始される場合

---

***inc***

プロミスカスモードオープナーの数をカウントします。

***disable***

デバイスが **promiscuity** モードを脱退する場合

---

**名前**

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

**概要**

```
netdev.transmit
```

**値*****protocol***

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

***length***

送信バッファーの長さ。

***true\_size***

送信されるデータのサイズ

***dev\_name***

デバイスの名前 (例: **eth0**、**ath1**)。

---

**名前**

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

**概要**

```
netdev.unregister
```

**値*****dev\_name***

登録解除されるデバイス

---

**名前**

`probe::netfilter.arp.forward` -- 転送される各 ARP パケットに対して呼び出されます。

## 概要

`netfilter.arp.forward`

## 値

### ***ar\_hln***

ハードウェアアドレスの長さ

### ***nf\_stop***

'stop' 判定を通知するために使用する定数

### ***outdev\_name***

パケットがルーティングされるネットワークデバイスの名前 (既知の場合)

### ***ar\_tha***

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

### ***nf\_accept***

'accept' 判定を通知するために使用する定数

### ***ar\_data***

ARP パケットデータリージョンのアドレス (ヘッダー後)

### ***indev\_name***

パケットが受信されたネットワークデバイスの名前 (既知の場合)

### ***arphdr***

ARP ヘッダーのアドレス

### ***outdev***

出力デバイスを表す `net_device` のアドレス (未知の場合は 0)

### ***nf\_repeat***

'repeat' 判定を通知するために使用する定数

### ***length***

パケットバッファの内容の長さ (バイト単位)

### ***nf\_stolen***

'stolen' 判定を通知するために使用する定数

### ***ar\_pln***

プロトコルアドレスの長さ

***pf***

プロトコルの種類 -- 常に「arp」

***ar\_sha***

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

***inde v***

入力デバイスを表す net\_device のアドレス (未知の場合は 0)

***nf\_drop***

'drop' 判定を通知するために使用する定数

***ar\_pro***

プロトコルアドレスの形式

***ar\_sip***

Ethernet+IP のみ (ar\_pro==0x800): ソース IP アドレス

***ar\_tip***

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

***ar\_hrd***

ハードウェアアドレスの形式

***nf\_queue***

'queue' 判定を通知するために使用する定数

***ar\_op***

ARP opcode (コマンド)

---

## 名前

probe::netfilter.arp.in -- 各受信 ARP パケットに対して呼び出されます。

## 概要

```
netfilter.arp.in
```

## 値

***ar\_hln***

ハードウェアアドレスの長さ

***nf\_stop***

'stop' 判定を通知するために使用する定数

***nf\_accept***

'accept' 判定を通知するために使用する定数

***ar\_tha***

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

***ar\_data***

ARP パケットデータリージョンのアドレス (ヘッダー後)

***outdev\_name***

パケットがルーティングされるネットワークデバイスの名前 (既知の場合)

***outdev***

出力デバイスを表す net\_device のアドレス (未知の場合は 0)

***nf\_repeat***

'repeat' 判定を通知するために使用する定数

***arphdr***

ARP ヘッダーのアドレス

***indev\_name***

パケットが受信されたネットワークデバイスの名前 (既知の場合)

***nf\_stolen***

'stolen' 判定を通知するために使用する定数

***length***

パケットバッファの内容の長さ (バイト単位)

***ar\_pln***

プロトコルアドレスの長さ

***ar\_sha***

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

***pf***

プロトコルの種類 -- 常に「arp」

***nf\_drop***

'drop' 判定を通知するために使用する定数

***ar\_pro***

プロトコルアドレスの形式

***ar\_sip***

Ethernet+IP のみ (ar\_pro==0x800): ソース IP アドレス

***indev***

入力デバイスを表す `net_device` のアドレス (未知の場合は 0)

***ar\_tip***

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

***ar\_hrd***

ハードウェアアドレスの形式

***ar\_op***

ARP opcode (コマンド)

***nf\_queue***

'queue' 判定を通知するために使用する定数

---

## 名前

`probe::netfilter.arp.out` -- 各送信 ARP パケットに対して呼び出されます。

## 概要

`netfilter.arp.out`

## 値

***ar\_tip***

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

***nf\_drop***

'drop' 判定を通知するために使用する定数

***ar\_pro***

プロトコルアドレスの形式

***ar\_sip***

Ethernet+IP のみ (ar\_pro==0x800): ソース IP アドレス

***indev***

入力デバイスを表す `net_device` のアドレス (未知の場合は 0)

***ar\_sha***

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

***pf***

プロトコルの種類 -- 常に「arp」

***ar\_op***

ARP opcode (コマンド)

***nf\_queue***

'queue' 判定を通知するために使用する定数

***ar\_hrd***

ハードウェアアドレスの形式

***nf\_accept***

'accept' 判定を通知するために使用する定数

***ar\_data***

ARP パケットデータリージョンのアドレス (ヘッダー後)

***ar\_tha***

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

***outdev\_name***

パケットがルーティングされるネットワークデバイスの名前 (既知の場合)

***nf\_stop***

'stop' 判定を通知するために使用する定数

***ar\_hln***

ハードウェアアドレスの長さ

***ar\_pln***

プロトコルアドレスの長さ

***nf\_stolen***

'stolen' 判定を通知するために使用する定数

***length***

パケットバッファの内容の長さ (バイト単位)

***outdev***

出力デバイスを表す net\_device のアドレス (未知の場合は 0)

***nf\_repeat***

'repeat' 判定を通知するために使用する定数

***arphdr***

ARP ヘッダーのアドレス

### ***indev\_name***

パケットが受信されたネットワークデバイスの名前 (既知の場合)

---

## 名前

`probe::netfilter.bridge.forward` – 他のコンピューター宛の受信ブリッジングパケットに対して呼び出されます。

## 概要

`netfilter.bridge.forward`

## 値

### ***br\_fd***

転送遅延 (1/256 秒)

### ***nf\_queue***

'queue' 判定を通知するために使用する定数

### ***brhdr***

ブリッジヘッダーのアドレス

### ***br\_mac***

ブリッジ MAC アドレス

### ***indev***

入力デバイスを表す `net_device` のアドレス (未知の場合は 0)

### ***br\_msg***

メッセージ期間 (1/256 秒)

### ***nf\_drop***

'drop' 判定を通知するために使用する定数

### ***llcproto\_stp***

ブリッジスパニングツリープロトコルパケットを通知するために使用する定数

### ***pf***

プロトコルの種類 -- 常に「bridge」。

### ***br\_vid***

プロトコルバージョン ID



***indev\_name***

パケットが受信されたネットワークデバイスの名前 (既知の場合)

***br\_poid***

ポート ID

***outdev***

出力デバイスを表す `net_device` のアドレス (未知の場合は 0)

***nf\_repeat***

'repeat' 判定を通知するために使用する定数

***llcpdu***

LLC プロトコルデータユニットのアドレス

***length***

パケットバッファの内容の長さ (バイト単位)

***nf\_stolen***

'stolen' 判定を通知するために使用する定数

***br\_cost***

送信元ブリッジからルートまでの総コスト

***nf\_stop***

'stop' 判定を通知するために使用する定数

***br\_type***

BPDU タイプ

***br\_max***

最大期間 (1/256 秒)

***br\_hitime***

Hello 時間 (1/256 秒)

***protocol***

パケットプロトコル

***br\_bid***

ブリッジの ID

***br\_rmac***

ルートブリッジ MAC アドレス

***br\_prid***

プロトコル ID

**outdev\_name**

パケットがルーティングされるネットワークデバイスの名前 (既知の場合)

**br\_flags**

BPDU フラグ

**nf\_accept**

'accept' 判定を通知するために使用する定数

**br\_rid**

ルートブリッジの ID

---

## 名前

`probe::netfilter.bridge.local_in` – ローカルコンピューター宛のブリッジングパケットに対して呼び出されます。

## 概要

```
netfilter.bridge.local_in
```

## 値

**llcproto\_stp**

ブリッジスパニングツリープロトコルパケットを通知するために使用する定数

**pf**

プロトコルの種類 -- 常に「bridge」。

**nf\_drop**

'drop' 判定を通知するために使用する定数

**br\_msg**

メッセージ期間 (1/256 秒)

**indev**

入力デバイスを表す `net_device` のアドレス (未知の場合は 0)

**nf\_queue**

'queue' 判定を通知するために使用する定数

**br\_fd**

転送遅延 (1/256 秒)

***br\_mac***

ブリッジ MAC アドレス

***brhdr***

ブリッジヘッダーのアドレス

***br\_rid***

ルートブリッジの ID

***nf\_accept***

'accept' 判定を通知するために使用する定数

***outdev\_name***

パケットがルーティングされるネットワークデバイスの名前 (既知の場合)

***br\_flags***

BPDU フラグ

***br\_prid***

プロトコル ID

***br\_hptime***

Hello 時間 (1/256 秒)

***protocol***

パケットプロトコル

***br\_bid***

ブリッジの ID

***br\_rmac***

ルートブリッジ MAC アドレス

***br\_max***

最大期間 (1/256 秒)

***br\_type***

BPDU タイプ

***nf\_stop***

'stop' 判定を通知するために使用する定数

***br\_cost***

送信元ブリッジからルートまでの総コスト

***nf\_stolen***

'stolen' 判定を通知するために使用する定数

***length***

パケットバッファーの内容の長さ (バイト単位)

***llcpdu***

LLC プロトコルデータユニットのアドレス

***outdev***

出力デバイスを表す `net_device` のアドレス (未知の場合は 0)

***nf\_repeat***

'repeat' 判定を通知するために使用する定数

***indev\_name***

パケットが受信されたネットワークデバイスの名前 (既知の場合)

***br\_poid***

ポート ID

***br\_vid***

プロトコルバージョン ID

---

## 名前

`probe::netfilter.bridge.local_out` – ローカルプロセスからのブリッジングパケットに対して呼び出されます。

## 概要

```
netfilter.bridge.local_out
```

## 値

***indev***

入力デバイスを表す `net_device` のアドレス (未知の場合は 0)

***br\_msg***

メッセージ期間 (1/256 秒)

***nf\_drop***

'drop' 判定を通知するために使用する定数

***llcproto\_stp***

ブリッジスパニングツリープロトコルパケットを通知するために使用する定数

***pf***

プロトコルの種類 -- 常に「bridge」。

***br\_fd***

転送遅延 (1/256 秒)

***nf\_queue***

'queue' 判定を通知するために使用する定数

***brhdr***

ブリッジヘッダーのアドレス

***br\_mac***

ブリッジ MAC アドレス

***br\_flags***

BPDU フラグ

***outdev\_name***

パケットがルーティングされるネットワークデバイスの名前 (既知の場合)

***nf\_accept***

'accept' 判定を通知するために使用する定数

***br\_rid***

ルートブリッジの ID

***nf\_stop***

'stop' 判定を通知するために使用する定数

***br\_type***

BPDU タイプ

***br\_max***

最大期間 (1/256 秒)

***protocol***

パケットプロトコル

***br\_hitime***

Hello 時間 (1/256 秒)

***br\_bid***

ブリッジの ID

***br\_rmac***

ルートブリッジ MAC アドレス

***br\_prid***

プロトコル ID

***llcpdu***

LLC プロトコルデータユニットのアドレス

***length***

パケットバッファの内容の長さ (バイト単位)

***nf\_stolen***

'stolen' 判定を通知するために使用する定数

***br\_cost***

送信元ブリッジからルートまでの総コスト

***br\_vid***

プロトコルバージョン ID

***indev\_name***

パケットが受信されたネットワークデバイスの名前 (既知の場合)

***br\_poid***

ポート ID

***outdev***

出力デバイスを表す `net_device` のアドレス (未知の場合は 0)

***nf\_repeat***

'repeat' 判定を通知するために使用する定数

---

## 名前

`probe::netfilter.bridge.post_routing` -- ブリッジングパケットがワイヤーに到達する前に呼び出されます。

## 概要

```
netfilter.bridge.post_routing
```

## 値

***llcproto\_stp***

ブリッジスパニングツリープロトコルパケットを通知するために使用する定数

***pf***

プロトコルの種類 -- 常に「bridge」。

***indev***

入力デバイスを表す `net_device` のアドレス (未知の場合は 0)

***nf\_drop***

'drop' 判定を通知するために使用する定数

***br\_msg***

メッセージ期間 (1/256 秒)

***nf\_queue***

'queue' 判定を通知するために使用する定数

***br\_mac***

ブリッジ MAC アドレス

***br\_fd***

転送遅延 (1/256 秒)

***brhdr***

ブリッジヘッダーのアドレス

***br\_hptime***

Hello 時間 (1/256 秒)

***br\_bid***

ブリッジの ID

***br\_rmac***

ルートブリッジ MAC アドレス

***protocol***

パケットプロトコル

***br\_prid***

プロトコル ID

***br\_type***

BPDU タイプ

***nf\_stop***

'stop' 判定を通知するために使用する定数

***br\_max***

最大期間 (1/256 秒)

***br\_rid***

ルートブリッジの ID

***br\_flags***

BPDU フラグ

***outdev\_name***

パケットがルーティングされるネットワークデバイスの名前 (既知の場合)

***nf\_accept***

'accept' 判定を通知するために使用する定数

***indev\_name***

パケットが受信されたネットワークデバイスの名前 (既知の場合)

***br\_poid***

ポート ID

***outdev***

出力デバイスを表す `net_device` のアドレス (未知の場合は 0)

***nf\_repeat***

'repeat' 判定を通知するために使用する定数

***br\_vid***

プロトコルバージョン ID

***length***

パケットバッファーの内容の長さ (バイト単位)

***nf\_stolen***

'stolen' 判定を通知するために使用する定数

***br\_cost***

送信元ブリッジからルートまでの総コスト

***llcpdu***

LLC プロトコルデータユニットのアドレス

---

名前



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

## 概要

`netfilter.bridge.pre_routing`

## 値

### ***llcproto\_stp***

ブリッジスパニングツリープロトコルパケットを通知するために使用する定数

### ***pf***

プロトコルの種類 -- 常に「**bridge**」。

### ***nf\_drop***

'drop' 判定を通知するために使用する定数

### ***br\_msg***

メッセージ期間 (1/256 秒)

### ***indev***

入力デバイスを表す `net_device` のアドレス (未知の場合は 0)

### ***brhdr***

ブリッジヘッダーのアドレス

### ***nf\_queue***

'queue' 判定を通知するために使用する定数

### ***br\_fd***

転送遅延 (1/256 秒)

### ***br\_mac***

ブリッジ MAC アドレス

### ***br\_rid***

ルートブリッジの ID

### ***nf\_accept***

'accept' 判定を通知するために使用する定数

### ***br\_flags***

BPDU フラグ

### ***outdev\_name***

パケットがルーティングされるネットワークデバイスの名前 (既知の場合)

***br\_prid***

プロトコル ID

***br\_rmac***

ルートブリッジ MAC アドレス

***br\_hptime***

Hello 時間 (1/256 秒)

***br\_bid***

ブリッジの ID

***protocol***

パケットプロトコル

***br\_max***

最大期間 (1/256 秒)

***br\_type***

BPDU タイプ

***nf\_stop***

'stop' 判定を通知するために使用する定数

***br\_cost***

送信元ブリッジからルートまでの総コスト

***nf\_stolen***

'stolen' 判定を通知するために使用する定数

***length***

パケットバッファの内容の長さ (バイト単位)

***llcpdu***

LLC プロトコルデータユニットのアドレス

***outdev***

出力デバイスを表す `net_device` のアドレス (未知の場合は 0)

***nf\_repeat***

'repeat' 判定を通知するために使用する定数

***indev\_name***

パケットが受信されたネットワークデバイスの名前 (既知の場合)

***br\_poid***

ポート ID

***br\_vid***

プロトコルバージョン ID

---

**名前**

`probe::netfilter.ip.forward` – 他のコンピューター宛の受信 IP パケットに対して呼び出されます。

**概要**

```
netfilter.ip.forward
```

**値*****saddr***

ソース IP アドレスを表す文字列

***sport***

TCP または UDP ソースポート (ipv4 のみ)

***daddr***

宛先 IP アドレスを表す文字列

***pf***

プロトコルの種類 -- 「ipv4」 または 「ipv6」

***indev***

入力デバイスを表す `net_device` のアドレス (未知の場合は 0)

***nf\_drop***

'drop' 判定を通知するために使用する定数

***nf\_queue***

'queue' 判定を通知するために使用する定数

***dport***

TCP または UDP 宛先ポート (ipv4 のみ)

***iphdr***

IP ヘッダーのアドレス

***fin***

TCP FIN フラグ (プロトコルが TCP の場合; ipv4 のみ)

***ack***

TCP ACK フラグ (プロトコルが TCP の場合; ipv4 のみ)

***syn***

TCP SYN フラグ (プロトコルが TCP の場合; ipv4 のみ)

***ipproto\_udp***

パケットプロトコルが UDP であることを通知するために使用する定数

***outdev\_name***

パケットがルーティングされるネットワークデバイスの名前 (既知の場合)

***nf\_accept***

'accept' 判定を通知するために使用する定数

***rst***

TCP RST フラグ (プロトコルが TCP の場合; ipv4 のみ)

***protocol***

ドライバーからのパケットプロトコル (ipv4 のみ)

***nf\_stop***

'stop' 判定を通知するために使用する定数

***length***

パケットバッファーの内容の長さ (バイト単位)

***nf\_stolen***

'stolen' 判定を通知するために使用する定数

***urg***

TCP URG フラグ (プロトコルが TCP の場合; ipv4 のみ)

***psh***

TCP PSH フラグ (プロトコルが TCP の場合; ipv4 のみ)

***ipproto\_tcp***

パケットプロトコルが TCP であることを通知するために使用する定数

***indev\_name***

パケットが受信されたネットワークデバイスの名前 (既知の場合)

***family***

IP アドレスの種類。

***outdev***

出力デバイスを表す `net_device` のアドレス (未知の場合は 0)

### ***nf\_repeat***

'repeat' 判定を通知するために使用する定数

---

## 名前

`probe::netfilter.ip.local_in` – ローカルコンピューター宛の受信 IP パケットに対して呼び出されます。

## 概要

`netfilter.ip.local_in`

## 値

### ***nf\_stolen***

'stolen' 判定を通知するために使用する定数

### ***length***

パケットバッファーの内容の長さ (バイト単位)

### ***urg***

TCP URG フラグ (プロトコルが TCP の場合; ipv4 のみ)

### ***psh***

TCP PSH フラグ (プロトコルが TCP の場合; ipv4 のみ)

### ***nf\_repeat***

'repeat' 判定を通知するために使用する定数

### ***family***

IP アドレスの種類。

### ***outdev***

出力デバイスを表す `net_device` のアドレス (未知の場合は 0)

### ***ipproto\_tcp***

パケットプロトコルが TCP であることを通知するために使用する定数

### ***indev\_name***

パケットが受信されたネットワークデバイスの名前 (既知の場合)

### ***nf\_accept***

'accept' 判定を通知するために使用する定数

***outdev\_name***

パケットがルーティングされるネットワークデバイスの名前 (既知の場合)

***protocol***

ドライバーからのパケットプロトコル (ipv4 のみ)

***rst***

TCP RST フラグ (プロトコルが TCP の場合; ipv4 のみ)

***nf\_stop***

'stop' 判定を通知するために使用する定数

***nf\_queue***

'queue' 判定を通知するために使用する定数

***dport***

TCP または UDP 宛先ポート (ipv4 のみ)

***iphdr***

IP ヘッダーのアドレス

***fin***

TCP FIN フラグ (プロトコルが TCP の場合; ipv4 のみ)

***syn***

TCP SYN フラグ (プロトコルが TCP の場合; ipv4 のみ)

***ack***

TCP ACK フラグ (プロトコルが TCP の場合; ipv4 のみ)

***ipproto\_udp***

パケットプロトコルが UDP であることを通知するために使用する定数

***saddr***

ソース IP アドレスを表す文字列

***sport***

TCP または UDP ソースポート (ipv4 のみ)

***pf***

プロトコルの種類 -- 「ipv4」 または 「ipv6」

***daddr***

宛先 IP アドレスを表す文字列

***nf\_drop***

'drop' 判定を通知するために使用する定数

### ***indev***

入力デバイスを表す `net_device` のアドレス (未知の場合は 0)

---

## 名前

`probe::netfilter.ip.local_out` – 送信 IP パケットに対して呼び出されます。

## 概要

`netfilter.ip.local_out`

## 値

### ***dport***

TCP または UDP 宛先ポート (ipv4 のみ)

### ***nf\_queue***

'queue' 判定を通知するために使用する定数

### ***syn***

TCP SYN フラグ (プロトコルが TCP の場合; ipv4 のみ)

### ***ipproto\_udp***

パケットプロトコルが UDP であることを通知するために使用する定数

### ***ack***

TCP ACK フラグ (プロトコルが TCP の場合; ipv4 のみ)

### ***fin***

TCP FIN フラグ (プロトコルが TCP の場合; ipv4 のみ)

### ***iphdr***

IP ヘッダーのアドレス

### ***saddr***

ソース IP アドレスを表す文字列

### ***sport***

TCP または UDP ソースポート (ipv4 のみ)

### ***indev***

入力デバイスを表す `net_device` のアドレス (未知の場合は 0)

***nf\_drop***

'drop' 判定を通知するために使用する定数

***daddr***

宛先 IP アドレスを表す文字列

***pf***

プロトコルの種類 -- 「ipv4」 または 「ipv6」

***psh***

TCP PSH フラグ (プロトコルが TCP の場合; ipv4 のみ)

***urg***

TCP URG フラグ (プロトコルが TCP の場合; ipv4 のみ)

***length***

パケットバッファの内容の長さ (バイト単位)

***nf\_stolen***

'stolen' 判定を通知するために使用する定数

***ipproto\_tcp***

パケットプロトコルが TCP であることを通知するために使用する定数

***indev\_name***

パケットが受信されたネットワークデバイスの名前 (既知の場合)

***nf\_repeat***

'repeat' 判定を通知するために使用する定数

***family***

IP アドレスの種類。

***outdev***

出力デバイスを表す `net_device` のアドレス (未知の場合は 0)

***outdev\_name***

パケットがルーティングされるネットワークデバイスの名前 (既知の場合)

***nf\_accept***

'accept' 判定を通知するために使用する定数

***nf\_stop***

'stop' 判定を通知するために使用する定数

***rst***



TCP RST フラグ (プロトコルが TCP の場合; ipv4 のみ)

### ***protocol***

ドライバーからのパケットプロトコル (ipv4 のみ)

---

## 名前

`probe::netfilter.ip.post_routing` – 送信 IP パケットがコンピューターから離れるとすぐに呼び出されます。

## 概要

```
netfilter.ip.post_routing
```

## 値

### ***family***

IP アドレスの種類。

### ***outdev***

出力デバイスを表す `net_device` のアドレス (未知の場合は 0)

### ***nf\_repeat***

'repeat' 判定を通知するために使用する定数

### ***ipproto\_tcp***

パケットプロトコルが TCP であることを通知するために使用する定数

### ***indev\_name***

パケットが受信されたネットワークデバイスの名前 (既知の場合)

### ***nf\_stolen***

'stolen' 判定を通知するために使用する定数

### ***length***

パケットバッファーの内容の長さ (バイト単位)

### ***urg***

TCP URG フラグ (プロトコルが TCP の場合; ipv4 のみ)

### ***psh***

TCP PSH フラグ (プロトコルが TCP の場合; ipv4 のみ)

### ***rst***

TCP RST フラグ (プロトコルが TCP の場合; ipv4 のみ)

***protocol***

ドライバーからのパケットプロトコル (ipv4 のみ)

***nf\_stop***

'stop' 判定を通知するために使用する定数

***nf\_accept***

'accept' 判定を通知するために使用する定数

***outdev\_name***

パケットがルーティングされるネットワークデバイスの名前 (既知の場合)

***iphdr***

IP ヘッダーのアドレス

***fin***

TCP FIN フラグ (プロトコルが TCP の場合; ipv4 のみ)

***syn***

TCP SYN フラグ (プロトコルが TCP の場合; ipv4 のみ)

***ipproto\_udp***

パケットプロトコルが UDP であることを通知するために使用する定数

***ack***

TCP ACK フラグ (プロトコルが TCP の場合; ipv4 のみ)

***nf\_queue***

'queue' 判定を通知するために使用する定数

***dport***

TCP または UDP 宛先ポート (ipv4 のみ)

***pf***

プロトコルの種類 -- 「ipv4」 または 「ipv6」

***daddr***

宛先 IP アドレスを表す文字列

***nf\_drop***

'drop' 判定を通知するために使用する定数

***indev***

入力デバイスを表す net\_device のアドレス (未知の場合は 0)

***saddr***

ソース IP アドレスを表す文字列

**sport**

TCP または UDP ソースポート (ipv4 のみ)

---

**名前**

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

**概要**

netfilter.ip.pre\_routing

**値****indev**

入力デバイスを表す net\_device のアドレス (未知の場合は 0)

**nf\_drop**

'drop' 判定を通知するために使用する定数

**daddr**

宛先 IP アドレスを表す文字列

**pf**

プロトコルの種類 - 'ipv4' または 'ipv6'。

**sport**

TCP または UDP ソースポート (ipv4 のみ)

**saddr**

ソース IP アドレスを表す文字列

**syn**

TCP SYN フラグ (プロトコルが TCP の場合; ipv4 のみ)

**ipproto\_udp**

パケットプロトコルが UDP であることを通知するために使用する定数

**ack**

TCP ACK フラグ (プロトコルが TCP の場合; ipv4 のみ)

**iphdr**

IP ヘッダーのアドレス

***fin***

TCP FIN フラグ (プロトコルが TCP の場合; ipv4 のみ)

***dport***

TCP または UDP 宛先ポート (ipv4 のみ)

***nf\_queue***

'queue' 判定を通知するために使用する定数

***nf\_stop***

'stop' 判定を通知するために使用する定数

***rst***

TCP RST フラグ (プロトコルが TCP の場合; ipv4 のみ)

***protocol***

ドライバーからのパケットプロトコル (ipv4 のみ)

***outdev\_name***

パケットがルーティングされるネットワークデバイスの名前 (既知の場合)

***nf\_accept***

'accept' 判定を通知するために使用する定数

***indev\_name***

パケットが受信されたネットワークデバイスの名前 (既知の場合)

***ipproto\_tcp***

パケットプロトコルが TCP であることを通知するために使用する定数

***family***

IP アドレスの種類。

***nf\_repeat***

'repeat' 判定を通知するために使用する定数

***outdev***

出力デバイスを表す `net_device` のアドレス (未知の場合は 0)

***psh***

TCP PSH フラグ (プロトコルが TCP の場合; ipv4 のみ)

***urg***

TCP URG フラグ (プロトコルが TCP の場合; ipv4 のみ)

***length***

パケットバッファの内容の長さ (バイト単位)

### ***nf\_stolen***

'stolen' 判定を通知するために使用する定数

---

## 名前

`probe::sunrpc.clnt.bind_new_program` – 新しい RPC プログラムを既存のクライアントにバインディングします。

## 概要

`sunrpc.clnt.bind_new_program`

## 値

### ***progrname***

新しい RPC プログラムの名前

### ***old\_prog***

古い RPC プログラムの番号

### ***vers***

新しい RPC プログラムのバージョン

### ***servername***

サーバーマシン名

### ***old\_vers***

古い RPC プログラムのバージョン

### ***old\_progrname***

古い RPC プログラムの名前

### ***prog***

新しい RPC プログラムの番号

---

## 名前

`probe::sunrpc.clnt.call_async` – 非同期 RPC 呼び出しを行います。

## 概要

`sunrpc.clnt.call_async`

## 値

**progname**

RPC プログラム名

**prot**

IP プロトコル番号

**proc**

この RPC 呼び出しのプロシーチャー番号

**procname**

この RPC 呼び出しのプロシーチャー名

**vers**

RPC プログラムバージョン番号

**flags**

フラグ

**servername**

サーバーマシン名

**xid**

現在の送信 ID

**port**

ポート番号

**prog**

RPC プログラム番号

**dead**

このクライアントを放棄するかどうかを指定します。

---

## 名前

probe::sunrpc.clnt.call\_sync – 同期 RPC 呼び出しを行います。

## 概要

`sunrpc.clnt.call_sync`

## 値

**vid**

---

**id**

現在の送信 ID

**servername**

サーバーマシン名

**flags**

フラグ

**dead**

このクライアントを放棄するかどうかを指定します。

**prog**

RPC プログラム番号

**port**

ポート番号

**prot**

IP プロトコル番号

**progrname**

RPC プログラム名

**vers**

RPC プログラムバージョン番号

**proc**

この RPC 呼び出しのプロシーチャー番号

**procname**

この RPC 呼び出しのプロシーチャー名

---

名前

probe::sunrpc.clnt.clone\_client – RPC クライアント構造をクローンします。

概要

sunrpc.clnt.clone\_client

値

**authflavor**

認証フレーバー

***port***

ポート番号

***progrname***

RPC プログラム名

***servername***

サーバーマシン名

***prot***

IP プロトコル番号

***prog***

RPC プログラム番号

***vers***

RPC プログラムバージョン番号

---

## 名前

probe::sunrpc.clnt.create\_client – RPC クライアントを作成します。

## 概要

```
sunrpc.clnt.create_client
```

## 値

***servername***

サーバーマシン名

***prot***

IP プロトコル番号

***authflavor***

認証フレーバー

***port***

ポート番号

***progrname***

RPC プログラム名

***vers***

RPC プログラムバージョン番号



**prog**

RPC プログラム番号

## 名前

probe::sunrpc.clnt.restart\_call – 非同期 RPC 呼び出しを再開します。

## 概要

**sunrpc.clnt.restart\_call**

## 値

**servername**

サーバーマシン名

**tk\_priority**

タスク優先度

**xid**

送信 ID

**prog**

RPC プログラム番号

**tk\_runstate**

タスク実行ステータス

**tk\_pid**

タスクのデバッグ支援

**ktk\_flags**

タスクフラグ

## 名前

probe::sunrpc.clnt.shutdown\_client – RPC クライアントをシャットダウンします。

## 概要

**sunrpc.clnt.shutdown\_client**

## 値

**om\_queue**

**om\_qmsgs**

xmit に対してキューに格納された jiffies

**clones**

クローンの数

**vers**

RPC プログラムバージョン番号

**om\_rtt**

RPC RTT jiffies

**om\_execute**

RPC 実行 jiffies

**rpccnt**

RPC 呼び出しの数

**progname**

RPC プログラム名

**authflavor**

認証フレーバー

**prot**

IP プロトコル番号

**prog**

RPC プログラム番号

**om\_bytes\_recv**

受信バイト数

**om\_bytes\_sent**

送信バイト数

**port**

ポート番号

**om\_ntrans**

RPC 送信の数

**netreconn**

再接続の数

**om\_ops**

操作の数

**tasks**

参照の数

**servername**

サーバーマシン名

---

## 名前

probe::sunrpc.sched.delay – RPC タスクを遅延します。

## 概要

`sunrpc.sched.delay`

## 値

**prog**

RPC 呼び出しのプログラム番号

**xid**

RPC 呼び出しの送信 ID

**delay**

遅延時間

**vers**

RPC 呼び出しのプログラムバージョン

**ktk\_flags**

タスクのフラグ

**tk\_pid**

タスクのデバッグ ID

**prot**

RPC 呼び出しの IP プロトコル

---

## 名前

probe::sunrpc.sched.execute – RPC `scheduler' を実行します。

## 概要

`sunrpc.sched.execute`

## 値

### ***tk\_pid***

タスクのデバッグ ID

### ***prot***

RPC 呼び出しの IP プロトコル

### ***vers***

RPC 呼び出しのプログラムバージョン

### ***ktk\_flags***

タスクのフラグ

### ***xid***

RPC 呼び出しの送信 ID

### ***prog***

RPC 呼び出しのプログラム番号

---

## 名前

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

## 概要

`sunrpc.sched.new_task`

## 値

### ***xid***

RPC 呼び出しの送信 ID

### ***prog***

RPC 呼び出しのプログラム番号

### ***prot***

RPC 呼び出しの IP プロトコル

### ***vers***

RPC 呼び出しのプログラムバージョン

### ***ktk\_flags***

## タスクのフラグ

---

### 名前

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

### 概要

`sunrpc.sched.release_task`

### 値

#### *prot*

RPC 呼び出しの IP プロトコル

#### *ktk\_flags*

タスクのフラグ

#### *vers*

RPC 呼び出しのプログラムバージョン

#### *xid*

RPC 呼び出しの送信 ID

#### *prog*

RPC 呼び出しのプログラム番号

### 説明

特定のカーネルの `rpc_release_task` 関数が見つからなかった可能性があります。見つからない場合は、すべてに対して `-1` を返します。

---

### 名前

`probe::sunrpc.svc.create` – RPC サービスを作成します。

### 概要

`sunrpc.svc.create`

### 値

#### *bufsize*

バッファサイズ

#### *pg\_nvers*

サポートされているバージョンの数

**progrname**

プログラムの名前

**prog**

プログラムの数

---

## 名前

probe::sunrpc.svc.destroy – RPC サービスを破棄します。

## 概要

```
sunrpc.svc.destroy
```

## 値

**sv\_nrthreads**

同時スレッドの数

**sv\_name**

サービス名

**sv\_prog**

プログラムの数

**rpcbadauth**

認証失敗のため破棄された要求の数

**rpcbadfmt**

不正な形式のため破棄された要求の数

**rpccnt**

実効 RPC 要求の数

**sv\_progrname**

プログラムの名前

**netcnt**

受信された RPC 要求の数

**nettcpconn**

受け入れられた TCP 接続の数

## 名前

probe::sunrpc.svc.drop – RPC 要求を破棄します。

## 概要

```
sunrpc.svc.drop
```

## 値

### ***rq\_xid***

要求の送信 ID。

### ***sv\_name***

サービス名

### ***rq\_prot***

要求の IP プロトコル。

### ***peer\_ip***

要求の送信元ピアアドレス。

### ***rq\_proc***

要求のプロシージャ番号。

### ***rq\_vers***

要求のプログラムバージョン。

### ***rq\_prog***

要求のプログラム番号。

## 名前

probe::sunrpc.svc.process – RPC 要求を処理します。

## 概要

```
sunrpc.svc.process
```

## 値

### ***rq\_prog***

要求のプログラム番号。

### ***rq\_vers***

要求のプログラムバージョン。

***peer\_ip***

要求の送信元ピアアドレス。

***rq\_proc***

要求のプロシーダ番号。

***sv\_prog***

プログラムの数

***rq\_prot***

要求の IP プロトコル。

***sv\_name***

サービス名

***rq\_xid***

要求の送信 ID。

***sv\_nthreads***

同時スレッドの数

---

## 名前

`probe::sunrpc.svc.recv` – 任意のソケットで次の RPC 要求をリッスンします。

## 概要

`sunrpc.svc.recv`

## 値

***sv\_nthreads***

同時スレッドの数

***sv\_name***

サービス名

***sv\_prog***

プログラムの数

***timeout***

データ待機のタイムアウト。



---

## 名前

probe::sunrpc.svc.register – ローカルポートマッパで RPC サービスを登録します。

## 概要

sunrpc.svc.register

## 値

### **sv\_name**

サービス名

### **prog**

プログラムの数

### **port**

ポート番号

### **progrname**

プログラムの名前

### **prot**

IP プロトコル番号

## 説明

**proto** と **port** は 0 であり、サービスを登録解除します。

---

## 名前

probe::sunrpc.svc.send – RPC クライアントへの返信を返します。

## 概要

sunrpc.svc.send

## 値

### **rq\_vers**

要求のプログラムバージョン。

### **rq\_prog**

要求のプログラム番号。

### **rq\_prot**

要求の IP プロトコル。

***sv\_name***

サービス名

***rq\_xid***

要求の送信 ID。

***peer\_ip***

要求の送信元ピアアドレス。

***rq\_proc***

要求のプロシージャ番号。

---

## 名前

`probe::tcp.disconnect` – TCP ソケットの接続解除。

## 概要

`tcp.disconnect`

## 値

***flags***

TCP フラグ (FIN など)。

***daddr***

宛先 IP アドレスを表す文字列

***sport***

TCP ソースポート。

***family***

IP アドレスの種類。

***name***

このプローブの名前。

***saddr***

ソース IP アドレスを表す文字列

***dport***

TCP 宛先ポート。

**sock**

ネットワークソケット。

**コンテキスト**

TCP を接続解除するプロセス。

---

**名前**

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

**概要**

```
tcp.disconnect.return
```

**値****name**

このプローブの名前。

**ret**

エラーコード (0: エラーなし)。

**コンテキスト**

TCP を接続解除するプロセス。

---

**名前**

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

**概要**

```
tcp.receive
```

**値****psh**

TCP PSH フラグ。

**ack**

TCP ACK フラグ。

**daddr**

宛先 IP アドレスを表す文字列

**syn**

TCP SYN フラグ。

***rst***

TCP RST フラグ。

***sport***

TCP ソースポート。

***protocol***

ドライバからのパケットプロトコル。

***urg***

TCP URG フラグ。

***name***

プローブポイントの名前。

***family***

IP アドレスの種類。

***fin***

TCP FIN フラグ。

***saddr***

ソース IP アドレスを表す文字列

***iphdr***

IP ヘッダーアドレス。

***dport***

TCP 宛先ポート。

---

## 名前

probe::tcp.recvmsg – TCP メッセージの受信。

## 概要

**|** tcp.recvmsg

## 値

***daddr***

宛先 IP アドレスを表す文字列

**sport**

TCP ソースポート。

**size**

受信バイト数。

**name**

このプローブの名前。

**family**

IP アドレスの種類。

**saddr**

ソース IP アドレスを表す文字列

**sock**

ネットワークソケット。

**dport**

TCP 宛先ポート。

## コンテキスト

TCP メッセージを受信するプロセス。

---

## 名前

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

## 概要

tcp.recvmsg.return

## 値

**saddr**

ソース IP アドレスを表す文字列

**dport**

TCP 宛先ポート。

**daddr**

宛先 IP アドレスを表す文字列

**size**

エラーが発生した場合の受信バイト数またはエラーコード。

**sport**

TCP ソースポート。

**family**

IP アドレスの種類。

**name**

このプローブの名前。

## コンテキスト

TCP メッセージを受信するプロセス。

---

## 名前

probe::tcp.sendmsg – TCP メッセージの送信。

## 概要

```
tcp.sendmsg
```

## 値

**family**

IP アドレスの種類。

**sock**

ネットワークソケット。

**name**

このプローブの名前。

**size**

送信バイト数。

## コンテキスト

TCP メッセージを送信するプロセス。

---

## 名前

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

## 概要

```
tcp.sendmsg.return
```

## 値

### ***name***

このプローブの名前。

### ***size***

エラーが発生した場合の送信バイト数またはエラーコード。

## コンテキスト

TCP メッセージを送信するプロセス。

---

## 名前

`probe::tcp.setsockopt` – `setsockopt` の呼び出し。

## 概要

`tcp.setsockopt`

## 値

### ***optstr***

`optname` を人間が判読できる形式に解決します。

### ***name***

このプローブの名前。

### ***family***

IP アドレスの種類。

### ***level***

ソケットオプションが処理されるレベル。

### ***optname***

TCP ソケットオプション (TCP\_NODELAY や TCP\_MAXSEG など)。

### ***sock***

ネットワークソケット。

### ***optlen***

`setsockopt` の値にアクセスするために使用されます。

## コンテキスト

`setsockopt` を呼び出すプロセス。

---

## 名前

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

## 概要

```
tcp.setsockopt.return
```

## 値

### *ret*

エラーコード (0: エラーなし)。

### *name*

このプローブの名前。

## コンテキスト

`setsockopt` を呼び出すプロセス。

---

## 名前

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

## 概要

```
udp.disconnect
```

## 値

### *daddr*

宛先 IP アドレスを表す文字列

### *sock*

プロセスにより使用されるネットワークソケット。

### *saddr*

ソース IP アドレスを表す文字列

### *sport*

UDP ソースポート。

### *flags*

フラグ (FIN など)。

### *dport*

UDP 宛先ポート。



***name***

このプローブの名前。

***family***

IP アドレスの種類。

## コンテキスト

UDP の接続解除を要求するプロセス。

---

## 名前

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

## 概要

```
udp.disconnect.return
```

## 値

***saddr***

ソース IP アドレスを表す文字列

***sport***

UDP ソースポート。

***dport***

UDP 宛先ポート。

***family***

IP アドレスの種類。

***name***

このプローブの名前。

***daddr***

宛先 IP アドレスを表す文字列

***ret***

エラーコード (0: エラーなし)。

## コンテキスト

UDP の接続解除を要求したプロセス

---

## 名前

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

## 概要

`udp.recvmsg`

## 値

### *size*

プロセスが受信したバイト数。

### *sock*

プロセスにより使用されるネットワークソケット。

### *daddr*

宛先 IP アドレスを表す文字列

### *family*

IP アドレスの種類。

### *name*

このプローブの名前。

### *dport*

UDP 宛先ポート。

### *saddr*

ソース IP アドレスを表す文字列

### *sport*

UDP ソースポート。

## コンテキスト

UDP メッセージを受信したプロセス。

---

## 名前

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

## 概要

`udp.recvmsg.return`

## 値

-----

***name***

このプローブの名前。

***family***

IP アドレスの種類。

***dport***

UDP 宛先ポート。

***saddr***

ソース IP アドレスを表す文字列

***sport***

UDP ソースポート。

***size***

プロセスが受信したバイト数。

***daddr***

宛先 IP アドレスを表す文字列

## コンテキスト

UDP メッセージを受信したプロセス。

---

## 名前

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

## 概要

`udp.sendmsg`

## 値

***daddr***

宛先 IP アドレスを表す文字列

***sock***

プロセスにより使用されるネットワークソケット。

***size***

プロセスが送信したバイト数。

***saddr***

ソース IP アドレスを表す文字列

**sport**

UDP ソースポート。

**family**

IP アドレスの種類。

**name**

このプローブの名前。

**dport**

UDP 宛先ポート。

## コンテキスト

UDP メッセージを送信したプロセス。

---

## 名前

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

## 概要

`udp.sendmsg.return`

## 値

**size**

プロセスが送信したバイト数。

**name**

このプローブの名前。

## コンテキスト

UDP メッセージを送信したプロセス。

## 第15章 ソケットタップセット

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

### 名前

**function::inet\_get\_ip\_source** – カーネルソケットの IP ソースアドレス文字列を提供します。

### 概要

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

### 引数

#### **sock**

カーネルソケットのポインター。

---

### 名前

**function::inet\_get\_local\_port** – カーネルソケットのローカルポート番号を提供します。

### 概要

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

### 引数

#### **sock**

カーネルソケットのポインター。

---

### 名前

**function::sock\_fam\_num2str** – プロトコルファミリー番号が指定される場合、文字列表示を返します。

### 概要

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

### 引数

#### **family**

ファミリー番号。

---

## 名前

**function::sock\_fam\_str2num** – プロトコルファミリー名 (文字列) が指定される場合、対応するプロトコルファミリー番号を返します。

## 概要

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

## 引数

### ***family***

ファミリー名。

---

## 名前

**function::sock\_prot\_num2str** – プロトコル番号が指定される場合、文字列表現を返します。

## 概要

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

## 引数

### ***proto***

プロトコル番号。

---

## 名前

**function::sock\_prot\_str2num** – プロトコル名 (文字列) が指定される場合、対応するプロトコル番号を返します。

## 概要

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

## 引数

### ***proto***

プロトコル名。

---

## 名前

**function::sock\_state\_num2str** – ソケット状態番号が指定される場合、文字列表現を返します。

## 概要

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

## 引数

### **state**

状態番号。

## 名前

**function::sock\_state\_str2num** – ソケット状態文字列が指定される場合、対応する状態番号を返します。

## 概要

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

## 引数

### **state**

状態名。

## 名前

**probe::socket.aio\_read** – **sock\_aio\_read** を介してメッセージを受信します。

## 概要

```
socket.aio_read
```

## 値

### **flags**

ソケットフラグ値。

### **type**

ソケットタイプの値。

### **size**

メッセージサイズ (バイト単位)。

### **family**

プロトコルファミリー値。

***name***

このプローブの名前。

***protocol***

プロトコル値。

***state***

ソケット状態値。

## コンテキスト

メッセージの送信者。

## 説明

**sock\_aio\_read** 関数を使ったソケットでのメッセージ受信の開始時に実行されます。

---

## 名前

probe::socket.aio\_read.return – **sock\_aio\_read** を使ったメッセージ受信の終了。

## 概要

```
socket.aio_read.return
```

## 値

***family***

プロトコルファミリー値。

***protocol***

プロトコル値。

***name***

このプローブの名前。

***state***

ソケット状態値。

***success***

正常に受信されたかどうか (1 = yes、0 = no)。

***flags***

ソケットフラグ値。

***type***

ソケットタイプの値。



**size**

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

**コンテキスト**

メッセージ受信者。

**説明**

**sock\_aio\_read** 関数を使ったソケットでのメッセージ受信の終了時に実行されます。

---

**名前**

**probe::socket.aio\_write – sock\_aio\_write** で送信されるメッセージ。

**概要**

```
socket.aio_write
```

**値****flags**

ソケットフラグ値。

**type**

ソケットタイプの値。

**size**

メッセージサイズ (バイト単位)。

**family**

プロトコルファミリー値。

**protocol**

プロトコル値。

**name**

このプローブの名前。

**state**

ソケット状態値。

**コンテキスト**

メッセージの送信者。

**説明**

**sock\_aio\_write** 関数を使ったソケットでのメッセージ送信の開始時に実行されます。

## 名前

`probe::socket.aio_write.return – sock_aio_write` でのメッセージ送信の終了。

## 概要

```
socket.aio_write.return
```

## 値

### **state**

ソケット状態値。

### **success**

正常に受信されたかどうか (1 = yes、0 = no)。

### **family**

プロトコルファミリー値。

### **protocol**

プロトコル値。

### **name**

このプローブの名前。

### **type**

ソケットタイプの値。

### **size**

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

### **flags**

ソケットフラグ値。

## コンテキスト

メッセージ受信者。

## 説明

`sock_aio_write` 関数を使ったソケットでのメッセージ送信の終了時に実行されます。

---

## 名前

`probe::socket.close` – ソケットを閉じます。

## 概要

`socket.close`

## 値

### *type*

ソケットタイプの値。

### *flags*

ソケットフラグ値。

### *state*

ソケット状態値。

### *family*

プロトコルファミリー値。

### *name*

このプローブの名前。

### *protocol*

プロトコル値。

## コンテキスト

リクエスター (ユーザープロセスまたはカーネル)。

## 説明

ソケットのクローズ開始時に実行されます。

---

## 名前

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

## 概要

`socket.close.return`

## 値

### *name*

このプローブの名前。

## コンテキスト

リクエスター (ユーザープロセスまたはカーネル)。

## 説明

ソケットのクローズ終了時に実行されます。

---

## 名前

`probe::socket.create` – ソケットの作成。

## 概要

`socket.create`

## 値

### *type*

ソケットタイプの値。

### *name*

このプローブの名前。

### *protocol*

プロトコル値。

### *family*

プロトコルファミリー値。

### *requester*

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

## コンテキスト

リクエスター (リクエスター変数を参照)。

## 説明

ソケットの作成開始時に実行されます。

---

## 名前

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

## 概要

`socket.create.return`

## 値

### *success*

ソケットが正常に作成されたかどうか (1 = yes、0 = no)。

### *family*

プロトコルファミリー値。

**requester**

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

**name**

このプローブの名前。

**protocol**

プロトコル値。

**type**

ソケットタイプの値。

**err**

`success == 0` の場合のエラーコード。

## コンテキスト

リクエスター (ユーザープロセスまたはカーネル)。

## 説明

ソケットの作成終了時に実行されます。

---

## 名前

`probe::socket.read_iter` – `sock_read_iter` を使ったメッセージの受信。

## 概要

`socket.read_iter`

## 値

**state**

ソケット状態値。

**protocol**

プロトコル値。

**name**

このプローブの名前。

**family**

プロトコルファミリー値。

**size**

メッセージサイズ (バイト単位)。

**type**

ソケットタイプの値。

**flags**

ソケットフラグ値。

## コンテキスト

メッセージの送信者。

## 説明

**sock\_read\_iter** 関数を使ったソケットでのメッセージ受信の開始時に実行されます。

---

## 名前

probe::socket.read\_iter.return – **sock\_read\_iter** を使ったメッセージ受信の終了。

## 概要

```
socket.read_iter.return
```

## 値

**flags**

ソケットフラグ値。

**type**

ソケットタイプの値。

**size**

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

**family**

プロトコルファミリー値。

**name**

このプローブの名前。

**protocol**

プロトコル値。

**state**

ソケット状態値。

**success**

正常に受信されたかどうか (1 = yes、0 = no)。

## コンテキスト

メッセージ受信者。

## 説明

**sock\_read\_iter** 関数を使ったソケットでのメッセージ受信の終了時に実行されます。

---

## 名前

probe::socket.readv – **sock\_readv** を使ったメッセージの受信。

## 概要

`socket.readv`

## 値

### **state**

ソケット状態値。

### **family**

プロトコルファミリー値。

### **protocol**

プロトコル値。

### **name**

このプローブの名前。

### **type**

ソケットタイプの値。

### **size**

メッセージサイズ (バイト単位)。

### **flags**

ソケットフラグ値。

## コンテキスト

メッセージの送信者。

## 説明

**sock\_readv** 関数を使ったソケットでのメッセージ受信の開始時に実行されます。

---

## 名前

`probe::socket.readv.return – sock_readv` を使ったメッセージ受信の終了。

## 概要

```
socket.readv.return
```

## 値

### *name*

このプローブの名前。

### *protocol*

プロトコル値。

### *family*

プロトコルファミリー値。

### *success*

正常に受信されたかどうか (1 = yes、0 = no)。

### *state*

ソケット状態値。

### *flags*

ソケットフラグ値。

### *size*

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

### *type*

ソケットタイプの値。

## コンテキスト

メッセージ受信者。

## 説明

`sock_readv` 関数を使ったソケットでのメッセージ受信の終了時に実行されます。

---

## 名前

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

## 概要

```
socket.receive
```



## 値

**name**

このプローブの名前。

**protocol**

プロトコル値。

**family**

プロトコルファミリー値。

**success**

正常に送信されたかどうか (1 = yes、0 = no)。

**state**

ソケット状態値。

**flags**

ソケットフラグ値。

**size**

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

**type**

ソケットタイプの値。

## コンテキスト

メッセージ受信者。

---

## 名前

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

## 概要

`socket.recvmsg`

## 値

**family**

プロトコルファミリー値。

**name**

このプローブの名前。

**protocol**

プロトコル値。

**state**

ソケット状態値。

**flags**

ソケットフラグ値。

**type**

ソケットタイプの値。

**size**

メッセージサイズ (バイト単位)。

## コンテキスト

メッセージ受信者。

## 説明

**sock\_recvmsg** 関数を使ったソケットでのメッセージ受信の開始時に実行されます。

---

## 名前

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

## 概要

```
socket.recvmsg.return
```

## 値

**family**

プロトコルファミリー値。

**name**

このプローブの名前。

**protocol**

プロトコル値。

**state**

ソケット状態値。

**success**

正常に受信されたかどうか (1 = yes、0 = no)。

**flags**

ソケットフラグ値。

**type**

ソケットタイプの値。

**size**

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

**コンテキスト**

メッセージ受信者。

**説明**

**sock\_recvmsg** 関数を使ったソケットでのメッセージ受信の終了時に実行されます。

---

**名前**

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

**概要**

`socket.send`

**値****flags**

ソケットフラグ値。

**size**

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

**type**

ソケットタイプの値。

**protocol**

プロトコル値。

**name**

このプローブの名前。

**family**

プロトコルファミリー値。

**success**

正常に送信されたかどうか (1 = yes、0 = no)。

**state**

ソケット状態値。

**コンテキスト**

メッセージの送信者。

---

**名前**

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

**概要**

`socket.sendmsg`

**値****family**

プロトコルファミリー値。

**name**

このプローブの名前。

**protocol**

プロトコル値。

**state**

ソケット状態値。

**flags**

ソケットフラグ値。

**type**

ソケットタイプの値。

**size**

メッセージサイズ (バイト単位)。

**コンテキスト**

メッセージの送信者。

**説明**

`sock_sendmsg` 関数を使ったソケットでのメッセージ送信の開始時に実行されます。

---

## 名前

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

## 概要

`socket.sendmsg.return`

## 値

### *type*

ソケットタイプの値。

### *size*

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

### *flags*

ソケットフラグ値。

### *state*

ソケット状態値。

### *success*

正常に送信されたかどうか (1 = yes、0 = no)。

### *family*

プロトコルファミリー値。

### *protocol*

プロトコル値。

### *name*

このプローブの名前。

## コンテキスト

メッセージの送信者。

## 説明

`sock_sendmsg` 関数を使ったソケットでのメッセージ送信の終了時に実行されます。

## 名前

`probe::socket.write_iter` – `sock_write_iter` で送信されるメッセージ。

## 概要

`socket.write_iter`

## 値

### ***state***

ソケット状態値。

### ***family***

プロトコルファミリー値。

### ***protocol***

プロトコル値。

### ***name***

このプローブの名前。

### ***type***

ソケットタイプの値。

### ***size***

メッセージサイズ (バイト単位)。

### ***flags***

ソケットフラグ値。

## コンテキスト

メッセージの送信者。

## 説明

**sock\_write\_iter** 関数を使ったソケットでのメッセージ送信の開始時に実行されます。

---

## 名前

`probe::socket.write_iter.return – sock_write_iter` でのメッセージ送信の終了。

## 概要

```
socket.write_iter.return
```

## 値

### ***type***

ソケットタイプの値。

### ***size***

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

### ***flags***

ソケットフラグ値。

**state**

ソケット状態値。

**success**

正常に受信されたかどうか (1 = yes、0 = no)。

**family**

プロトコルファミリー値。

**protocol**

プロトコル値。

**name**

このプローブの名前。

## コンテキスト

メッセージ受信者。

## 説明

**sock\_write\_iter** 関数を使ったソケットでのメッセージ送信の終了時に実行されます。

---

## 名前

probe::socket.writev – **socket\_writev** で送信されるメッセージ。

## 概要

**socket.writev**

## 値

**state**

ソケット状態値。

**protocol**

プロトコル値。

**name**

このプローブの名前。

**family**

プロトコルファミリー値。

**size**

メッセージサイズ (バイト単位)。

**type**

ソケットタイプの値。

**flags**

ソケットフラグ値。

**コンテキスト**

メッセージの送信者。

**説明**

**sock\_writev** 関数を使ったソケットでのメッセージ送信の開始時に実行されます。

---

**名前**

**probe::socket.writev.return – socket\_writev** でのメッセージ送信の終了。

**概要**

```
socket.writev.return
```

**値****success**

正常に送信されたかどうか (1 = yes、0 = no)。

**state**

ソケット状態値。

**name**

このプローブの名前。

**protocol**

プロトコル値。

**family**

プロトコルファミリー値。

**size**

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

**type**

ソケットタイプの値。

**flags**



ソケットフラグ値。

コンテキスト  
メッセージ受信者。

## 説明

**sock\_writev** 関数を使ったソケットでのメッセージ送信の終了時に実行されます。

## 第16章 SNMP 情報タップセット

この種類のプローブポイントは、SNMP タイプ情報を提供するためにソケットの活動をプローブするために使用されます。以下の関数とプローブポイントが含まれます。

### 名前

function::ipmib\_filter\_key – ipmib.\* プローブのデフォルトフィルター関数。

### 概要

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

### 引数

#### **skb**

構造 `sk_buff` のポインター。

#### **op**

**skb** がフィルターを通過した場合にカウントされる値。

#### **SourceIsLocal**

1 はローカル操作であり、0 は非ローカル操作です。

### 説明

この関数はデフォルトのフィルター関数です。ユーザーはこの関数を独自の関数に置き換えることができます。ユーザーにより提供されるフィルター関数は、**skb** の値に基づいてインデックスキーを返します。戻り値が 0 の場合、この特定の **skb** はカウントされません。

---

### 名前

function::ipmib\_get\_proto – プロトコル値を取得します。

### 概要

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

### 引数

#### **skb**

構造 `sk_buff` のポインター。

### 説明

**skb** からプロトコル値を返します。

---

## 名前

function::ipmib\_local\_addr – ローカル IP アドレスを取得します。

## 概要

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

## 引数

### **skb**

構造 **sk\_buff** のポインター。

### **SourceIsLocal**

ローカル操作かどうかを示すフラグ。

## 説明

ローカル IP アドレス **skb** を返します。

---

## 名前

function::ipmib\_remote\_addr – リモート IP アドレスを取得します。

## 概要

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

## 引数

### **skb**

構造 **sk\_buff** のポインター。

### **SourceIsLocal**

ローカル操作かどうかを示すフラグ。

## 説明

**skb** からリモート IP アドレスを返します。

---

## 名前

function::ipmib\_tcp\_local\_port – ローカル TCP ポートを取得します。

## 概要

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

## 引数

### **skb**

構造 `sk_buff` のポインター。

### **SourceIsLocal**

ローカル操作かどうかを示すフラグ。

## 説明

**skb** からローカル TCP ポートを返します。

---

## 名前

function::ipmib\_tcp\_remote\_port – リモート TCP ポートを取得します。

## 概要

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

## 引数

### **skb**

構造 `sk_buff` のポインター。

### **SourceIsLocal**

ローカル操作かどうかを示すフラグ。

## 説明

**skb** からリモート TCP ポートを返します。

---

## 名前

function::linuxmib\_filter\_key – linuxmib.\* プローブのデフォルトフィルター関数。

## 概要

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

## 引数

### **sk**

構造 `sock` のポインター。

### **op**

**sk** がフィルターを通過する場合にカウントされる値。

---

## 説明

この関数はデフォルトのフィルター関数です。ユーザーはこの関数を独自の関数に置き換えることができます。ユーザーにより提供されるフィルター関数は、**sk**の値に基づいてインデックスキーを返します。戻り値が0の場合、この特定の**sk**はカウントされません。

---

## 名前

function::tcpmib\_filter\_key – tcpmib.\* プロープのデフォルトフィルター関数。

## 概要

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

## 引数

### sk

処理される構造 **sock** のポインター。

### op

**sk** がフィルターを通過する場合にカウントされる値。

## 説明

この関数はデフォルトのフィルター関数です。ユーザーはこの関数を独自の関数に置き換えることができます。ユーザーにより提供されるフィルター関数は、**sk**の値に基づいてインデックスキーを返します。戻り値が0の場合、この特定の**sk**はカウントされません。

---

## 名前

function::tcpmib\_get\_state – ソケットの状態を取得します。

## 概要

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

## 引数

### sk

構造 **sock** のポインター。

## 説明

構造 **sock** から **sk\_state** を返します。

---

## 名前

function::tcpmib\_local\_addr – ソースアドレスを取得します。

## 概要

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

## 引数

**sk**

構造 `inet_sock` のポインター。

## 説明

構造 `inet_sock` から `saddr` をホストの順序で返します。

---

## 名前

`function::tcpmib_local_port` – ローカルポートを取得します。

## 概要

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

## 引数

**sk**

構造 `inet_sock` のポインター。

## 説明

構造 `inet_sock` から `sport` をホストの順序で返します。

---

## 名前

`function::tcpmib_remote_addr` – リモートアドレスを取得します。

## 概要

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

## 引数

**sk**

構造 `inet_sock` のポインター。

## 説明

構造 `inet_sock` から `daddr` をホストの順序で返します。

---

## 名前

`function::tcpmib_remote_port` – リモートポートを取得します。

## 概要

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

## 引数

### *sk*

構造 `inet_sock` のポインター。

## 説明

構造 `inet_sock` から `dport` をホストの順序で返します。

---

## 名前

`probe::ipmib.ForwDatagrams` – 転送済みパケットをカウントします。

## 概要

```
ipmib.ForwDatagrams
```

## 値

### *op*

カウンターに追加する値 (デフォルト値は 1)。

### *skb*

処理される構造 `sk_buff` のポインター。

## 説明

*skb* により参照されるパケットは、関数 `ipmib_filter_key` によりフィルタリングされます。パケットはフィルターを通過すると、グローバル *ForwDatagrams* (SNMP の MIB `IPSTATS_MIB_OUTFORWDATAGRAMS` と同等) でカウントされます。

---

## 名前

`probe::ipmib.FragFails` – 断片化に失敗したデータグラムをカウントします。

## 概要

```
ipmib.FragFails
```

## 値

---

**op**

カウンターに追加する値 (デフォルト値は 1)。

**skb**

処理される構造 `sk_buff` のポインター。

**説明**

**skb** により参照されるパケットは、関数 `ipmib_filter_key` によりフィルタリングされます。パケットはフィルターを通過すると、グローバル *FragFails* (SNMP の MIB IPSTATS\_MIB\_FRAGFAILS と同等) でカウントされます。

---

**名前**

`probe::ipmib.FragOKs` – 正常に断片化されたデータグラムをカウントします。

**概要**

`ipmib.FragOKs`

**値****skb**

処理される構造 `sk_buff` のポインター。

**op**

カウンターに追加する値 (デフォルト値は 1)。

**説明**

**skb** により参照されるパケットは、関数 `ipmib_filter_key` によりフィルタリングされます。パケットはフィルターを通過すると、グローバル *FragOKs* (SNMP の MIB IPSTATS\_MIB\_FRAGOKS と同等) でカウントされます。

---

**名前**

`probe::ipmib.InAddrErrors` – 正しくないアドレスの受信パケットをカウントします。

**概要**

`ipmib.InAddrErrors`

**値****skb**

処理される構造 `sk_buff` のポインター。

**op**



カウンターに追加する値 (デフォルト値は 1)。

## 説明

**skb** により参照されるパケットは、関数 **ipmib\_filter\_key** によりフィルタリングされます。パケットはフィルターを通過すると、グローバル **InAddrErrors** (SNMP の MIB IPSTATS\_MIB\_INADDRERRORS と同等) でカウントされます。

---

## 名前

probe::ipmib.InDiscards – 破棄された受信パケットをカウントします。

## 概要

`ipmib.InDiscards`

## 値

### op

カウンターに追加する値 (デフォルト値は 1)。

### skb

処理される構造 **sk\_buff** のポインター。

## 説明

**skb** により参照されるパケットは、関数 **ipmib\_filter\_key** によりフィルタリングされます。パケットはフィルターを通過すると、グローバル **InDiscards** (SNMP の MIB STATS\_MIB\_INDISCARDS と同等) でカウントされます。

---

## 名前

probe::ipmib.InNoRoutes – 一致するソケットがない受信パケットをカウントします。

## 概要

`ipmib.InNoRoutes`

## 値

### op

カウンターに追加する値 (デフォルト値は 1)。

### skb

処理される構造 **sk\_buff** のポインター。

## 説明

**skb** により参照されるパケットは、関数 **ipmib\_filter\_key** によりフィルタリングされます。パケットはフィルターを通過すると、グローバル **InNoRoutes** (SNMP の MIB IPSTATS\_MIB\_INNOROUTES と同等) でカウントされます。

---

## 名前

probe::ipmib.InReceives – 受信パケットをカウントします。

## 概要

```
ipmib.InReceives
```

## 値

### skb

処理される構造 **sk\_buff** のポインター。

### op

カウンターに追加する値 (デフォルト値は 1)。

## 説明

**skb** により参照されるパケットは、関数 **ipmib\_filter\_key** によりフィルタリングされます。パケットはフィルターを通過すると、グローバル **InReceives** (SNMP の MIB IPSTATS\_MIB\_INRECEIVES と同等) でカウントされます。

---

## 名前

probe::ipmib.InUnknownProtos – proto がバインドされていない受信パケットをカウントします。

## 概要

```
ipmib.InUnknownProtos
```

## 値

### skb

処理される構造 **sk\_buff** のポインター。

### op

カウンターに追加する値 (デフォルト値は 1)。

## 説明

**skb** により参照されるパケットは、関数 **ipmib\_filter\_key** によりフィルタリングされます。パケットはフィルターを通過すると、グローバル **InUnknownProtos** (SNMP の MIB IPSTATS\_MIB\_INUNKNOWNPROTOS と同等) でカウントされます。

---

## 名前

probe::ipmib.OutRequests – パケットを送信する要求をカウントします。

## 概要

`ipmib.OutRequests`

## 値

### *skb*

処理される構造 `sk_buff` のポインター。

### *op*

カウンターに追加する値 (デフォルト値は 1)。

## 説明

*skb* により参照されるパケットは、関数 `ipmib_filter_key` によりフィルタリングされます。パケットはフィルターを通過すると、グローバル *OutRequests* (SNMP の MIB IPSTATS\_MIB\_OUTREQUESTS と同等) でカウントされます。

---

## 名前

probe::ipmib.ReasmReqds – パケット断片化再構築要求の数をカウントします。

## 概要

`ipmib.ReasmReqds`

## 値

### *op*

カウンターに追加する値 (デフォルト値は 1)。

### *skb*

処理される構造 `sk_buff` のポインター。

## 説明

*skb* により参照されるパケットは、関数 `ipmib_filter_key` によりフィルタリングされます。パケットはフィルターを通過すると、グローバル *ReasmReqds* (SNMP の MIB IPSTATS\_MIB\_REASMREQDS と同等) でカウントされます。

---

## 名前

probe::ipmib.ReasmTimeout – 再構築タイムアウトをカウントします。

## 概要

`ipmib.ReasmTimeout`

## 値

### *op*

カウンターに追加する値 (デフォルト値は 1)。

### *skb*

処理される構造 `sk_buff` のポインター。

## 説明

*skb* により参照されるパケットは、関数 `ipmib_filter_key` によりフィルタリングされます。パケットはフィルターを通過すると、グローバル *ReasmTimeout* (SNMP の MIB IPSTATS\_MIB\_REASMTIMEOUT と同等) でカウントされます。

---

## 名前

probe::linuxmib.DelayedACKs – 遅延 ack をカウントします。

## 概要

`linuxmib.DelayedACKs`

## 値

### *op*

カウンターに追加する値 (デフォルト値は 1)。

### *sk*

処理される `struct sock` のポインター。

## 説明

*skb* により参照されるパケットは、関数 `linuxmib_filter_key` によりフィルタリングされます。パケットはフィルターを通過すると、グローバル *DelayedACKs* (SNMP の MIB LINUX\_MIB\_DELAYEDACKS と同等) でカウントされます。

---

## 名前

probe::linuxmib.ListenDrops – 破棄された times conn 要求の数。

## 概要

`linuxmib.ListenDrops`

## 値

**sk**

処理される **struct sock** のポインター。

**op**

カウンターに追加する値 (デフォルト値は 1)。

## 説明

**skb** により参照されるパケットは、関数 **linuxmib\_filter\_key** によりフィルタリングされます。パケットはフィルターを通過すると、グローバル **ListenDrops** (SNMP の MIB **LINUX\_MIB\_LISTENDROPS** と同等) でカウントされます。

---

## 名前

probe::linuxmib.ListenOverflows – リッスンキューがオーバーフローした回数。

## 概要

```
linuxmib.ListenOverflows
```

## 値

**sk**

処理される **struct sock** のポインター。

**op**

カウンターに追加する値 (デフォルト値は 1)。

## 説明

**skb** により参照されるパケットは、関数 **linuxmib\_filter\_key** によりフィルタリングされます。パケットはフィルターを通過すると、グローバル **ListenOverflows** (SNMP の MIB **LINUX\_MIB\_LISTENOVERFLOWS** と同等) でカウントされます。

---

## 名前

probe::linuxmib.TCPMemoryPressures – メモリープレッシャーが使用された回数。

## 概要

```
linuxmib.TCPMemoryPressures
```

## 値

**sk**

処理される **struct sock** のポインター。

**op**

カウンターに追加する値 (デフォルト値は 1)。

## 説明

**skb** により参照されるパケットは、関数 **linuxmib\_filter\_key** によりフィルタリングされます。パケットはフィルターを通過すると、グローバル **TCPMemoryPressures** (SNMP の MIB **LINUX\_MIB\_TCPMEMORYPRESSURES** と同等) でカウントされます。

---

## 名前

**probe::tcpmib.ActiveOpens** – アクティブで開いているソケットをカウントします。

## 概要

**tcpmib.ActiveOpens**

## 値

### op

カウンターに追加する値 (デフォルト値は 1)。

### sk

処理される構造 **sock** のポインター。

## 説明

**skb** により参照されるパケットは、関数 **tcpmib\_filter\_key** によりフィルタリングされます。パケットはフィルターを通過すると、グローバル **ActiveOpens** (SNMP の MIB **TCP\_MIB\_ACTIVEOPENS** と同等) でカウントされます。

---

## 名前

**probe::tcpmib.AttemptFails** – ソケットを開くのに失敗した回数をカウントします。

## 概要

**tcpmib.AttemptFails**

## 値

### op

カウンターに追加する値 (デフォルト値は 1)。

### sk

処理される構造 **sock** のポインター。

## 説明

**skb** により参照されるパケットは、関数 **tcpmib\_filter\_key** によりフィルタリングされます。パケットはフィルターを通過すると、グローバル **AttemptFails** (SNMP の MIB TCP\_MIB\_ATTEMPTFAILS と同等) でカウントされます。

---

## 名前

**probe::tcpmib.CurrEstab** – オープン状態のソケットの数を更新します。

## 概要

**tcpmib.CurrEstab**

## 値

### sk

処理される構造 **sock** のポインター。

### op

カウンターに追加する値 (デフォルト値は 1)。

## 説明

**skb** により参照されるパケットは、関数 **tcpmib\_filter\_key** によりフィルタリングされます。パケットはフィルターを通過すると、グローバル **CurrEstab** (SNMP の MIB TCP\_MIB\_CURRESTAB と同等) でカウントされます。

---

## 名前

**probe::tcpmib.EstabResets** – ソケットのリセットをカウントします。

## 概要

**tcpmib.EstabResets**

## 値

### sk

処理される構造 **sock** のポインター。

### op

カウンターに追加する値 (デフォルト値は 1)。

## 説明

**skb** により参照されるパケットは、関数 **tcpmib\_filter\_key** によりフィルタリングされます。パケットはフィルターを通過すると、グローバル **EstabResets** (SNMP の MIB TCP\_MIB\_ESTABRESETS と同等) でカウントされます。

---

## 名前

`probe::tcpmib.InSegs` – 受信 `tcp` セグメントをカウントします。

## 概要

`tcpmib.InSegs`

## 値

### *op*

カウンターに追加する値 (デフォルト値は 1)。

### *sk*

処理される構造 `sock` のポインター。

## 説明

*skb* により参照されるパケットは、関数 `tcpmib_filter_key` (`tcp v4` の場合は `ipmib_filter_key`) によりフィルタリングされます。パケットはフィルターを通過すると、グローバル *InSegs* (SNMP の MIB TCP\_MIB\_INSEGS と同等) でカウントされます。

---

## 名前

`probe::tcpmib.OutRsts` – リセットパケットの送信をカウントします。

## 概要

`tcpmib.OutRsts`

## 値

### *sk*

処理される構造 `sock` のポインター。

### *op*

カウンターに追加する値 (デフォルト値は 1)。

## 説明

*skb* により参照されるパケットは、関数 `tcpmib_filter_key` によりフィルタリングされます。パケットはフィルターを通過すると、グローバル *OutRsts* (SNMP の MIB TCP\_MIB\_OUTRSTS と同等) でカウントされます。

---

## 名前

`probe::tcpmib.OutSegs` – TCP セグメントの送信をカウントします。



## 概要

`tcpmib.OutSegs`

## 値

### *sk*

処理される構造 `sock` のポインター。

### *op*

カウンターに追加する値 (デフォルト値は 1)。

## 説明

*skb* により参照されるパケットは、関数 `tcpmib_filter_key` によりフィルタリングされます。パケットはフィルターを通過すると、グローバル *OutSegs* (SNMP の MIB TCP\_MIB\_OUTSEGS と同等) でカウントされます。

---

## 名前

`probe::tcpmib.PassiveOpens` – ソケットのパッシブな作成をカウントします。

## 概要

`tcpmib.PassiveOpens`

## 値

### *sk*

処理される構造 `sock` のポインター。

### *op*

カウンターに追加する値 (デフォルト値は 1)。

## 説明

*skb* により参照されるパケットは、関数 `tcpmib_filter_key` によりフィルタリングされます。パケットはフィルターを通過すると、グローバル *PassiveOpens* (SNMP の MIB TCP\_MIB\_PASSIVEOPENS と同等) でカウントされます。

---

## 名前

`probe::tcpmib.RetransSegs` – TCP セグメントの再送信をカウントします。

## 概要

`tcpmib.RetransSegs`

## 値

### *op*

カウンターに追加する値 (デフォルト値は 1)。

### *sk*

処理される構造 **sock** のポインター。

## 説明

**skb** により参照されるパケットは、関数 **tcpmib\_filter\_key** によりフィルタリングされます。パケットはフィルターを通過すると、グローバル **RetransSegs** (SNMP の MIB **TCP\_MIB\_RETRANSSEGS** と同等) でカウントされます。

## 第17章 カーネルプロセススタップセット

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

### 名前

**function::get\_loadavg\_index** – 指定された間隔の負荷平均値を取得します。

### 概要

```
get_loadavg_index:long(indx:long)
```

### 引数

#### **indx**

取得する負荷平均間隔。

### 説明

この関数は、指定された間隔で負荷平均値を返します。3つの負荷平均値1分、5分、および15分は `avenrun` アレイのインデックス 0、1、および 2 に対応します (`linux/sched.h` を参照)。負荷平均値の切り捨て整数部分が返されることに注意してください。指定されたインデックスが範囲外の場合は、エラーメッセージと例外が発生します。

### 名前

**function::sprint\_loadavg** – プリティープリントの負荷平均値を報告します。

### 概要

```
sprint_loadavg:string()
```

### 引数

なし

### 説明

1分、5分、および15分の負荷平均値の通常の形式で、3桁の10進数値とともに文字列を返します。

### 名前

**function::target\_set\_pid** – PID はターゲットプロセスに基づきますか？

### 概要

```
target_set_pid(pid:)
```

### 引数

***pid***

問い合わせるプロセスのPID。

**説明**

この関数は、該当する **process-id** が「**target set**」の範囲内であるかどうか (つまり、**id** が最上位の **target** プロセスの子孫であるかどうか) を返します。

---

**名前**

**function::target\_set\_report** – ターゲットセットに関するレポートを出力します。

**概要**

```
target_set_report()
```

**引数**

なし

**説明**

この関数は、ターゲットセットのプロセスと先祖に関するレポートを出力します。

---

**名前**

**probe::kprocess.create** – 新規プロセスまたはスレッドが正常に作成されたときに必ず実行されます。

**概要**

```
kprocess.create
```

**値*****new\_tid***

新規に作成されたタスクの TID。

***new\_pid***

新規に作成されたプロセスの PID。

**コンテキスト**

作成されたプロセスの親。

**説明**

フォーク (またはシステムコールのいずれか) または新しいカーネルスレッドの結果、新規プロセスが正常に作成されたときに必ず実行されます。

---

## 名前

probe::kprocess.exec – 新しいプログラムの実行の試行。

## 概要

`kprocess.exec`

## 値

### **filename**

新しい実行可能ファイルのパス。

### **name**

システムコールの名前 (「`execve`」) (SystemTap v2.5+)。

### **args**

0 番目の `arg` を含む、新しい実行可能ファイルに渡す引数 (SystemTap v2.5+)。

### **argstr**

0 番目の `arg` を除く、ファイル名の後に渡す引数が指定された文字列 (SystemTap v2.5+)。

## コンテキスト

`exec` の呼び出し元。

## 説明

プロセスが新しいプログラムを実行しようとしたときに必ず実行されます。SystemTap v2.5+ の `syscall.execve` プローブにエイリアスされます。

---

## 名前

probe::kprocess.exec\_complete – 新しいプログラムの実行から返されます。

## 概要

`kprocess.exec_complete`

## 値

### **retstr**

`errno` 文字列の表現 (SystemTap v2.5+)。

### **success**

実行が成功したかどうかを示すブール値。

### **errno**

実行の結果返されたエラー番号。

***name***

システムコールの名前 (「`execve`」) (SystemTap v2.5+)。

**コンテキスト**

成功すると、新しい実行可能ファイルのコンテキストになります。失敗すると、呼び出し元のコンテキストのままになります。

**説明**

実行呼び出しの完了時に実行されます。SystemTap v2.5+ の `syscall.execve.return` プローブにエイリアスされます。

---

**名前**

`probe::kprocess.exit` – プロセスを終了します。

**概要**

`kprocess.exit`

**値*****code***

プロセスの終了コード。

**コンテキスト**

終了するプロセス。

**説明**

プロセスが終了するときに実行されます。この後で `kprocess.release` が常に実行されます。ただし、プロセスがゾンビ状態で待機している場合、`kprocess.release` の実行は遅延することがあります。

---

**名前**

`probe::kprocess.release` – リリースされるプロセス。

**概要**

`kprocess.release`

**値*****released\_tid***

リリースされるタスクの TID。

***task***

リリースされるプロセスのタスクハンドル。

***released\_pid***

リリースされるプロセスの PID。

***pid***

互換性を維持するために ***released\_pid*** と同じ (非推奨)。

**コンテキスト**

親のコンテキスト (このプロセスの終了を通知したい場合。それ以外の場合は、プロセス自体のコンテキスト)。

**説明**

プロセスがカーネルからリリースされたときに実行されます。この後で **kprocess.exit** が必ず実行されます。ただし、プロセスがゾンビ状態で待機している場合、**kprocess.exit** の実行は遅延することがあります。

**名前**

**probe::kprocess.start** – 新規プロセスを開始します。

**概要**

**kprocess.start**

**値**

なし

**コンテキスト**

新規に作成されたプロセス。

**説明**

新規プロセスが実行を開始する直前に実行されます。

## 第18章 シグナルタップセット

この種類のプローブポイントは、シグナル活動をプローブするために使用されます。以下のプローブポイントが含まれます。

### 名前

**function::get\_sa\_flags** – sa\_flags の数値を返します。

### 概要

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

### 引数

#### **act**

問い合わせる **sigaction** のアドレス。

---

### 名前

**function::get\_sa\_handler** – sa\_handler の数値を返します。

### 概要

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

### 引数

#### **act**

問い合わせる **sigaction** のアドレス。

---

### 名前

**function::is\_sig\_blocked** – シグナルが現在ブロックされている場合に 1 を返し、ブロックされていない場合に 0 を返します。

### 概要

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

### 引数

#### **task**

問い合わせる **task\_struct** のアドレス。

#### **sig**



テストするシグナル番号。

---

## 名前

**function::sa\_flags\_str** – **sa\_flags** の文字列表現を返します。

## 概要

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

## 引数

### **sa\_flags**

文字列に変換するフラグのセット。

---

## 名前

**function::sa\_handler\_str** – **sa\_handler** の文字列表現を返します。

## 概要

```
sa_handler_str(handler:)
```

## 引数

### **handler**

文字列に変換する **sa\_handler**。

## 説明

**sa\_handler** の文字列表現を返します。**SIG\_DFL**、**SIG\_IGN**、または **SIG\_ERR** ではない場合は、ハンドラーのアドレスを返します。

---

## 名前

**function::signal\_str** – シグナル番号の文字列表現を返します。

## 概要

```
signal_str(num:)
```

## 引数

### **num**

文字列に変換するシグナル番号。

---

## 名前

**function::sigset\_mask\_str** – **sigset** の文字列表現を返します。

## 概要

```
sigset_mask_str:string(mask:long)
```

## 引数

### **mask**

文字列に変換する **sigset**。

---

## 名前

**probe::signal.check\_ignored** – シグナルが無視されたことを確認します。

## 概要

```
signal.check_ignored
```

## 値

### **sig\_pid**

シグナルを受信するプロセスの PID。

### **sig**

シグナルの番号。

### **sig\_name**

シグナルの文字列表現。

### **pid\_name**

シグナルを受信するプロセスの名前

---

## 名前

**probe::signal.check\_ignored.return** – シグナルが無視されたことの確認が完了しました。

## 概要

```
signal.check_ignored.return
```

## 値

***name***

プローブポイントの名前。

***retstr***

値を文字列として返します。

---

**名前**

`probe::signal.checkperm` – 送信されたシグナルについての確認が実行されます。

**概要**

`signal.checkperm`

**値*****pid\_name***

シグナルを受信するプロセスの名前

***task***

シグナル受信者のタスクハンドル。

***sig\_name***

シグナルの文字列表現。

***sinfo***

`siginfo` 構造のアドレス。

***name***

プローブポイントの名前。

***sig***

シグナルの番号。

***si\_code***

シグナルタイプを示します。

***sig\_pid***

シグナルを受信するプロセスの PID。

---

**名前**

`probe::signal.checkperm.return` – 送信されたシグナルの確認実行が完了しました。

## 概要

`signal.checkperm.return`

## 値

### ***retstr***

値を文字列として返します。

### ***name***

プローブポイントの名前。

---

## 名前

`probe::signal.do_action` – シグナルアクションを調査または変更します。

## 概要

`signal.do_action`

## 値

### ***sigact\_addr***

シグナルに関連付けられた新しい `sigaction` 構造のアドレス。

### ***sig\_name***

シグナルの文字列表現。

### ***sa\_mask***

シグナルの新しいマスク。

### ***sa\_handler***

シグナルの新しいハンドラー。

### ***oldsigact\_addr***

シグナルに関連付けられた古い `sigaction` 構造のアドレス。

### ***sig***

調査または変更するシグナル。

### ***name***

プローブポイントの名前。

---

## 名前

`probe::signal.do_action.return` – シグナルアクションの調査または変更が完了しました。

## 概要

```
signal.do_action.return
```

## 値

### ***retstr***

値を文字列として返します。

### ***name***

プローブポイントの名前。

---

## 名前

`probe::signal.flush` – タスクのすべての保留中シグナルを破棄します。

## 概要

```
signal.flush
```

## 値

### ***task***

破棄を実行するプロセスのタスクハンドラー。

### ***pid\_name***

破棄を実行するタスクに関連付けられたプロセスの名前。

### ***name***

プローブポイントの名前。

### ***sig\_pid***

破棄を実行するタスクに関連付けられたプロセスの PID。

---

## 名前

`probe::signal.force_segv` – SIGSEGV の送信を強制実行します。

## 概要

```
signal.force_segv
```

## 値

### ***sig\_name***

シグナルの文字列表現。

### ***pid\_name***

シグナルを受信するプロセスの名前

### ***sig\_pid***

シグナルを受信するプロセスの PID。

### ***name***

プローブポイントの名前。

### ***sig***

シグナルの番号。

---

## 名前

`probe::signal.force_segv.return` – SIGSEGV の送信の強制実行が完了しました。

## 概要

```
signal.force_segv.return
```

## 値

### ***retstr***

値を文字列として返します。

### ***name***

プローブポイントの名前。

---

## 名前

`probe::signal.handle` – 呼び出されるシグナルハンドラー。

## 概要

```
signal.handle
```

## 値

### ***name***

プローブポイントの名前。

***sig***

シグナルハンドラーを呼び出したシグナル番号。

***siginfo***

siginfo テーブルのアドレス。

***ka\_addr***

シグナルに関連付けられた **k\_sigaction** テーブルのアドレス。

***sig\_mode***

シグナルがユーザーモードシグナルであるか、またはカーネルモードシグナルであることを示します。

***sig\_code***

siginfo シグナルの **si\_code** 値。

***regs***

カーネルモードスタック領域のアドレス (**SystemTap 2.1** では非推奨)。

***oldset\_addr***

ブロックされたシグナルのビットマスクアレイのアドレス (**SystemTap 2.1** では非推奨)。

***sig\_name***

シグナルの文字列表現。

---

## 名前

**probe::signal.handle.return** – シグナルハンドラーの呼び出しが完了しました。

## 概要

**signal.handle.return**

## 値

***retstr***

値を文字列として返します。

***name***

プローブポイントの名前。

## 説明

(**SystemTap 2.1** では非推奨)。

## 名前

`probe::signal.pending` – 保留中シグナルの調査。

## 概要

```
signal.pending
```

## 値

### *name*

プローブポイントの名前。

### *sigset\_size*

ユーザー空間シグナルセットのサイズ。

### *sigset\_add*

ユーザー空間シグナルセットのアドレス (`sigset_t`)。

## 説明

このプローブは、特定のスレッドへの配信を待機しているシグナルのセットを調査するために使用されます。これは、通常 `do_sigpending` カーネル関数が実行されたときに行われます。

---

## 名前

`probe::signal.pending.return` – 保留中のシグナルの調査が完了しました。

## 概要

```
signal.pending.return
```

## 値

### *name*

プローブポイントの名前。

### *retstr*

値を文字列として返します。

---

## 名前

`probe::signal.procmask` – ブロックされたシグナルを調査または変更します。

## 概要



```
signal.procmask
```

## 値

### **name**

プローブポイントの名前。

### **sigset**

sigset\_t に設定する実際の値 (正しいか?)。

### **how**

ブロックされたシグナルを変更する方法を示します。値は SIG\_BLOCK=0 (シグナルをブロックする場合)、SIG\_UNBLOCK=1 (シグナルをブロック解除する場合)、および SIG\_SETMASK=2 (シグナルマスクを設定する場合) です。

### **sigset\_addr**

実装するシグナルセット (sigset\_t) のアドレス。

### **oldsigset\_addr**

シグナルセット (sigset\_t) の古いアドレス。

## 名前

probe::signal.procmask.return – ブロックされたシグナルの調査または変更が完了しました。

## 概要

```
signal.procmask.return
```

## 値

### **retstr**

値を文字列として返します。

### **name**

プローブポイントの名前。

## 名前

probe::signal.send – プロセスに送信されるシグナル。

## 概要

```
signal.send
```

## 値

### ***send2queue***

シグナルが既存の **sigqueue** に送信されたかどうかを示します (SystemTap 2.1 では非推奨)。

### ***pid\_name***

シグナル受信者の名前。

### ***task***

シグナル受信者のタスクハンドル。

### ***sig\_name***

シグナルの文字列表現。

### ***sinfo***

siginfo 構造のアドレス

### ***shared***

シグナルがスレッドグループで共有されているかどうかを示します。

### ***si\_code***

シグナルタイプを示します。

### ***name***

シグナルを送信するために使用される関数の名前。

### ***sig***

シグナルの番号。

### ***sig\_pid***

シグナルを受信するプロセスの PID。

## コンテキスト

シグナルの送信者。

---

## 名前

**probe::signal.send.return** – プロセスへのシグナルの送信が完了しました (SystemTap 2.1 では非推奨)。

## 概要

**signal.send.return**

## 値

### ***shared***

送信されたシグナルがスレッドグループで共有されているかどうかを示します。

### **name**

シグナルを送信するために使用される関数の名前。

### **retstr**

`__group_send_sig_info`、`specific_send_sig_info`、または `send_sigqueue` への戻り値。

### **send2queue**

送信されたシグナルが既存の `sigqueue` に送信されたかどうかを示します。

## コンテキスト

シグナルの送信者 (正しいか?)。

## 説明

`__group_send_sig_info` と `specific_send_sig_info` の戻り値は以下のとおりです。

0 -- シグナルがプロセスに正常に送信されました。つまり、(1) シグナルが受信側プロセスによって無視されました。(2) これは非 RT シグナルであり、システムのキューにはすでに 1 つのシグナルが格納されています。(3) シグナルが受信側プロセスの `sigqueue` に正常に追加されました。

-EAGAIN -- 受信側プロセスの `sigqueue` はオーバーフロー状態です。シグナルは RT であり、`kill` 以外の関数を使用しているユーザーによって送信されました。

`send_group_sigqueue` と `send_sigqueue` の戻り値は以下のとおりです。

0 -- シグナルは受信側プロセスの `sigqueue` に正常に追加されました。または、`SI_TIMER` エントリーがすでにキューに格納されています (この場合、オーバーランした数は単純に増分されます)。

1 -- シグナルが受信側プロセスによって無視されました。

-1 -- (`send_sigqueue` のみ) タスクは終了中とマークされ、`*posix_timer_event` をグループリーダーにリダイレクトすることが許可されます。

## 名前

`probe::signal.send_sig_queue` – シグナルをプロセスのキューに格納します。

## 概要

```
signal.send_sig_queue
```

## 値

### **sig**

キューに格納されたシグナル。

### **name**

プローブポイントの名前。

***sig\_pid***

シグナルがキューに格納されるプロセスの PID。

***pid\_name***

シグナルがキューに格納されるプロセスの名前。

***sig\_name***

シグナルの文字列表現。

***sigqueue\_addr***

シグナルキューのアドレス。

---

## 名前

`probe::signal.send_sig_queue.return` – プロセスのキューへのシグナルの格納が完了しました。

## 概要

```
signal.send_sig_queue.return
```

## 値

***retstr***

値を文字列として返します。

***name***

プローブポイントの名前。

---

## 名前

`probe::signal.sys_tgkill` – スレッドグループに kill シグナルを送信します。

## 概要

```
signal.sys_tgkill
```

## 値

***sig\_pid***

kill シグナルを受信するスレッドの PID。

***sig***

プロセスに送信される特定の kill シグナル。

***name***

プローブポイントの名前。

***pid\_name***

シグナル受信者の名前。

***sig\_name***

シグナルの文字列表現。

***tgid***

kill シグナルを受信するスレッドのスレッドグループ ID。

***task***

シグナル受信者のタスクハンドル。

## 説明

**tgkill** 呼び出しは **tkill** に似ています。ただし、呼び出し元は、シグナルを送信するスレッドのスレッドグループ ID を指定できます。これにより、TID の再使用を回避できます。

---

## 名前

**probe::signal.sys\_tgkill.return** – スレッドグループへの kill シグナルの送信が完了しました。

## 概要

```
signal.sys_tgkill.return
```

## 値

***name***

プローブポイントの名前。

***retstr***

**\_\_group\_send\_sig\_info** に対する戻り値。

---

## 名前

**probe::signal.sys\_tkill** – スレッドに kill シグナルを送信します。

## 概要

```
signal.sys_tkill
```

## 値

***sig\_pid***

kill シグナルを受信するプロセスの PID。

***sig***

プロセスに送信される特定のシグナル。

***name***

プローブポイントの名前。

***pid\_name***

シグナル受信者の名前。

***sig\_name***

シグナルの文字列表現。

***task***

シグナル受信者のタスクハンドル。

## 説明

tkill 呼び出しは kill(2) に似ています。ただし、特定のスレッドグループ内のプロセスを対象にすることができます。このようなプロセスは一意的スレッド ID (TID) を介して対象になります。

---

## 名前

probe::signal.syskill – プロセスに kill シグナルを送信します。

## 概要

```
signal.syskill
```

## 値

***sig\_pid***

シグナルを受信するプロセスの PID。

***sig***

プロセスに送信される特定のシグナル。

***name***

プローブポイントの名前。

***pid\_name***

シグナル受信者の名前。

***sig\_name***

シグナルの文字列表現。

**task**

シグナル受信者のタスクハンドル。

---

**名前**

probe::signal.syskill.return – kill シグナルの送信が完了しました。

**概要**

signal.syskill.return

**値**

なし

---

**名前**

probe::signal.systkill.return – スレッドへの kill シグナルの送信が完了しました。

**概要**

signal.systkill.return

**値****retstr**

\_\_group\_send\_sig\_info に対する戻り値。

**name**

プローブポイントの名前。

---

**名前**

probe::signal.wakeup – シグナルによりウェイクするスリープ状態のプロセス。

**概要**

signal.wakeup

**値****pid\_name**

ウェイクするプロセスの名前。

---

***resume***

STOPPED または TRACED 状態のタスクをウェイクアップするかどうかを示します。

***state\_mask***

ウェイクするタスク状態のマスクを示す文字列表現。可能な値は TASK\_INTERRUPTIBLE、TASK\_STOPPED、TASK\_TRACED、TASK\_WAKEKILL、および TASK\_INTERRUPTIBLE です。

***sig\_pid***

ウェイクするプロセスの PID。



## 第19章 ERRNO タップセット

この関数セットは、`errno` 数値を処理するために使用されます。以下の関数が含まれます。

### 名前

`function::errno_str` – エラーコードに関連付けられたシンボリック文字列。

### 概要

```
errno_str:string(err:long)
```

### 引数

#### *err*

受信されたエラー番号。

### 説明

この関数は、提供者エラーコードに関連付けられたシンボリック文字列を返します (番号が 2 の場合は `ENOENT`、または `3333` などの範囲外の値の場合は `E#3333`)。

---

### 名前

`function::return_str` – 戻り値を文字列としてフォーマットします。

### 概要

```
return_str:string(format:long,ret:long)
```

### 引数

#### *format*

戻り値の型の基準値を決定する変数。

#### *ret*

戻り値 (一般的には `$return`)。

### 説明

この関数は `syscall` タップセットで使用され、文字列を返します。書式を 10 進数の場合は 1、16 進数の場合は 2、8 進数の場合は 3 に設定します。

この関数は `returnstr` よりも優先されることに注意してください。

---

### 名前

`function::returnstr` – 戻り値を文字列としてフォーマットします。

## 概要

```
returnstr:string(format:long)
```

## 引数

### *format*

戻り値の型の基準値を決定する変数。

## 説明

この関数は `nd_syscall` タップセットで使用され、文字列を返します。書式を 10 進数の場合は 1、16 進数の場合は 2、8 進数の場合は 3 に設定します。

この関数は `dwarfless` プローブでのみ使用してください ('`kprobe.function(「foo」)` )。他のプローブは `return_str` を使用します。

---

## 名前

`function::returnval` – プローブされた関数の戻り値。

## 概要

```
returnval:long()
```

## 引数

なし

## 説明

一般的に関数の値が返されるレジスタの値を返します。**`$return`** が利用できないプローブで使用できます。これは、実際の戻り値の推定値にすぎず、完全に間違っていることがあります。通常は、`dwarfless` プローブでのみ使用されます。

## 第20章 RLIMIT タップセット

この関数セットは、リソース制限 (RLIMIT\_\*) を定義する文字列を処理するために使用され、リソース制限の対応する数値を返します。この関数セットには、以下の関数が含まれます。

### 名前

**function::rlimit\_from\_str** – リソース制限コードに関連付けられたシンボリック文字列。

### 概要

```
rlimit_from_str:long(lim_str:string)
```

### 引数

***lim\_str***

制限の文字列表現。

### 説明

この関数は、該当する文字列に関連付けられた数値を返します (文字列 **RLIMIT\_CPU** の場合は **0**、範囲外の値の場合は **-1** など)。

## 第21章 デバイスタップセット

この関数セットは、カーネルおよびユーザー空間デバイス番号を処理するために使用されます。以下の関数が含まれます。

### 名前

**function::MAJOR** – カーネルデバイス番号からメジャーデバイス番号を抽出します (*kdev\_t*)。

### 概要

```
MAJOR:long(dev:long)
```

### 引数

#### *dev*

問い合わせるカーネルデバイス番号。

---

### 名前

**function::MINOR** – カーネルデバイス番号からマイナーデバイス番号を抽出します (*kdev\_t*)。

### 概要

```
MINOR:long(dev:long)
```

### 引数

#### *dev*

問い合わせるカーネルデバイス番号。

---

### 名前

**function::MKDEV** – カーネルデバイス番号と比較できる値を作成します (*kdev\_t*)。

### 概要

```
MKDEV:long(major:long,minor:long)
```

### 引数

#### *major*

使用するメジャーデバイス番号。

#### *minor*

使用するマイナーデバイス番号。

---

## 名前

**function::usrdev2kerndev** – ユーザー空間デバイス番号をカーネルで使用されている形式に変換します。

## 概要

```
usrdev2kerndev:long(dev:long)
```

## 引数

### ***dev***

ユーザー空間形式のデバイス番号。

## 第22章 DIRECTORY-ENTRY (DENTRY) タップセット

この種類の関数は、ファイルまたは完全パス名にカーネル VFS ディレクトリーのエン트리ポインターをマップするために使用されます。

### 名前

function::d\_name – dirent 名を取得します。

### 概要

```
d_name:string(dentry:long)
```

### 引数

#### *dentry*

dentry へのポインター。

### 説明

dirent 名 (パスベース名) を返します。

---

### 名前

function::d\_path – 完全 nameidata パスを取得します。

### 概要

```
d_path:string(nd:long)
```

### 引数

#### *nd*

nameidata へのポインター。

### 説明

kernel d\_path 関数などの完全 dirent 名を返します (root への完全パス)。

---

### 名前

function::fullpath\_struct\_file – 完全パスを取得します。

### 概要

```
fullpath_struct_file:string(task:long,file:long)
```

### 引数

**task**

task\_struct ポインター。

**file**

「struct file」へのポインター。

**説明**

kernel d\_path 関数などの完全 dirent 名を返します (root への完全パス)。

---

**名前**

function::fullpath\_struct\_nameidata – 完全 nameidata パスを取得します。

**概要**

```
fullpath_struct_nameidata(nd:)
```

**引数****nd**

「struct nameidata」へのポインター。

**説明**

kernel (および systemtap-tapset) d\_path 関数などの、「/」の付いた完全 dirent 名を返します (root への完全パス)。

---

**名前**

function::fullpath\_struct\_path – 完全パスを取得します。

**概要**

```
fullpath_struct_path:string(path:long)
```

**引数****path**

「struct path」へのポインター。

**説明**

kernel d\_path 関数などの完全 dirent 名を返します (root への完全パス)。

---

**名前**

**function::inode\_name** – inode 名を取得します。

## 概要

```
inode_name:string(inode:long)
```

## 引数

### *inode*

inode へのポインター。

## 説明

指定の **inode** に関連付けられた最初のパスのベース名を返します。

---

## 名前

**function::inode\_path** – inode へのパスを取得します。

## 概要

```
inode_path:string(inode:long)
```

## 引数

### *inode*

inode へのポインター。

## 説明

指定の **inode** に関連付けられた完全パスを返します。

---

## 名前

**function::real\_mount** – 'struct mount' ポインターを取得します。

## 概要

```
real_mount:long(vfsmnt:long)
```

## 引数

### *vfsmnt*

'struct vfsmount' のポインター。

## 説明

'struct vfsmount' ポインターの 'struct mount' ポインター値を返します。

---



---

## 名前

**function::reverse\_path\_walk** – 完全 dirent パスを取得します。

## 概要

```
reverse_path_walk:string(dentry:long)
```

## 引数

### **dentry**

dentry へのポインター。

## 説明

パス名を返します (マウントポイントへの部分パス)。

---

## 名前

**function::task\_dentry\_path** – 完全 dentry パスを取得します。

## 概要

```
task_dentry_path:string(task:long,dentry:long,vfsmnt:long)
```

## 引数

### **task**

task\_struct ポインター。

### **dentry**

dirent ポインター。

### **vfsmnt**

vfsmnt ポインター。

## 説明

kernel d\_path 関数などの完全 dirent 名を返します (root への完全パス)。

## 第23章 ロギングタップセット

この種類の関数は、単純なメッセージ文字列を各種の宛先に送信するために使用されます。

### 名前

**function::assert** – アサーションの評価。

### 概要

```
assert(expression:,msg:)
```

### 引数

#### *expression*

評価する式。

#### *msg*

フォーマットされたメッセージ文字列。

### 説明

この関数は式をチェックし、式がゼロと評価された場合に、現在実行されているプローブを終了します。**error** を使用します。**try{} catch{}** によって補足することができます。

---

### 名前

**function::error** – エラーメッセージを送信します。

### 概要

```
error(msg:string)
```

### 引数

#### *msg*

フォーマットされたメッセージ文字列。

### 説明

暗黙的な行末が追加されました。**staprun** は先頭に文字列「**ERROR:**」を追加します。エラーメッセージを送信すると、現在実行されているプローブが停止します。**MAXERRORS** パラメーターに応じて、**exit** がトリガーされることがあります。

---

### 名前

**function::exit** – プローブスクリプトのシャットダウンを開始します。

## 概要

```
exit()
```

## 引数

なし

## 説明

これは、スクリプトのシャットダウンを開始する要求をキューに格納します。新しいプローブは実行されませんが(「end」プローブを除く)、現在実行されているすべてのプローブは処理を完了することがあります。

---

## 名前

**function::ftrace** – **ftrace** リングバッファにメッセージを送信します。

## 概要

```
ftrace(msg:string)
```

## 引数

**msg**

フォーマットされたメッセージ文字列。

## 説明

**ftrace** リングバッファが設定され、利用可能な場合は、`/debugfs/tracing/trace` でメッセージを確認します。それ以外の場合、メッセージは破棄されることがあります。暗黙的な行末が追加されます。

---

## 名前

**function::log** – 共通トレースバッファに行を送信します。

## 概要

```
log(msg:string)
```

## 引数

**msg**

フォーマットされたメッセージ文字列。

## 説明

この関数はデータをログに記録します。ログはメッセージをすぐに **staprun** と一括トランスポート (relays) (使用される場合) に送信します。該当する最後の文字が改行でない場合は、改行が追加されます。この関数は **printf** ほど効率的ではなく、緊急メッセージにのみ使用します。

---

## 名前

**function::printk** – メッセージをカーネルトレースバッファーに送信します。

## 概要

```
printk(level:long,msg:string)
```

## 引数

### *level*

重大度レベルの整数 (0=KERN\_EMERG ... 7=KERN\_DEBUG)。

### *msg*

フォーマットされたメッセージ文字列。

## 説明

該当する重大度とともにテキストの行をカーネル **dmesg/console** に出力します。暗黙的な行末が追加されます。この関数は、すべてのカーネルプローブコンテキストから安全に呼び出すことができないことがあります。したがって、**guru** モードのみに制限されます。

---

## 名前

**function::warn** – 警告ストリームに行を送信します。

## 概要

```
warn(msg:string)
```

## 引数

### *msg*

フォーマットされたメッセージ文字列。

## 説明

この関数は、警告メッセージをすぐに **staprun** に送信します。警告メッセージは、一括トランスポート (**relays**) (使用される場合) を介しても送信されます。最後の文字が改行でない場合は、改行が追加されます。

## 第24章 キューに関連する統計タップセット

この種類の関数は、キューに関連するシステムのパフォーマンスを追跡するために使用されます。

### 名前

**function::qs\_done** – 要求の終了を記録する関数。

### 概要

```
qs_done(qname:string)
```

### 引数

#### **qname**

終了したサービスの名前。

### 説明

この関数は、該当するキューからの要求の処理が完了したことを記録します。

---

### 名前

**function::qs\_run** – 処理される待機キューの移動を記録する関数。

### 概要

```
qs_run(qname:string)
```

### 引数

#### **qname**

移動および開始するサービスの名前。

### 説明

この関数は、以前にキューに格納された要求が該当するキューから削除され、現在処理されていることを記録します。

---

### 名前

**function::qs\_wait** – 要求のキューへの格納を記録する関数。

### 概要

```
qs_wait(qname:string)
```

### 引数

**qname**

キューへの格納を要求するキューの名前。

**説明**

この関数は、該当するキュー名に対して新しい要求がキューに格納されたことを記録します。

---

**名前**

**function::qsq\_blocked** – 要求が待機キューに存在した時間を返します。

**概要**

```
qsq_blocked:long(qname:string,scale:long)
```

**引数****qname**

キューの名前。

**scale**

時間の一部を考慮する変数。

**説明**

この関数は、1つまたは複数の要求が待機キューに存在した経過時間の一部を返します。

---

**名前**

**function::qsq\_print** – 該当するキューの統計の行を出力します。

**概要**

```
qsq_print(qname:string)
```

**引数****qname**

キューの名前。

**説明**

この関数は以下のものを含む行を出力します。

**該当するキューの統計**

キューの名前、1秒あたりの平均要求数、平均待機キュー時間、待機キューの平均時間、要求を処理する平均時間、待機キューが使用された時間の割合、および要求が処理された時間の割合。

---

## 名前

`function::qsq_service_time` – 1つの要求サービスあたりの時間。

## 概要

```
qsq_service_time:long(qname:string,scale:long)
```

## 引数

### *qname*

キューの名前。

### *scale*

時間の一部を考慮する変数。

## 説明

この関数は、要求が待機キューから削除された場合に、要求を処理するのに必要な時間の平均値 (マイクロ秒単位) を返します。

---

## 名前

`function::qsq_start` – キューの統計をリセットする関数。

## 概要

```
qsq_start(qname:string)
```

## 引数

### *qname*

終了したサービスの名前。

## 説明

この関数は、該当するキューの統計カウンターをリセットし、関数が呼び出されたときから追跡を再開します。この関数は、キューを作成および初期化するためにも使用されます。

---

## 名前

`function::qsq_throughput` – 1つのユニット時間あたりの処理済み要求数。

## 概要

```
qsq_throughput:long(qname:string,scale:long)
```

## 引数

### *qname*

***qname***

キューの名前。

***scale***

時間の一部を考慮する変数。

## 説明

この関数は、1 マイクロ秒ごとに処理された要求の平均数を返します。

---

## 名前

`function::qsq_utilization` – 要求が処理された時間の一部。

## 概要

```
qsq_utilization:long(qname:string, scale:long)
```

## 引数

***qname***

キューの名前。

***scale***

時間の一部を考慮する変数。

## 説明

この関数は、少なくとも 1 つの要求が処理された時間の平均値 (マイクロ秒単位) を返します。

---

## 名前

`function::qsq_wait_queue_length` – 待機キューの長さ。

## 概要

```
qsq_wait_queue_length:long(qname:string, scale:long)
```

## 引数

***qname***

キューの名前。

***scale***

時間の一部を考慮する変数。

## 説明



この関数は、待機キューの平均的な長さを返します。

---

## 名前

`function::qsq_wait_time` – 1つの要求あたりのキューおよびサービスの時間。

## 概要

```
qsq_wait_time:long(qname:string,scale:long)
```

## 引数

### ***qname***

キューの名前。

### ***scale***

時間の一部を考慮する変数。

## 説明

この関数は、要求の処理にかかる平均的な時間 (マイクロ秒単位) を返します (`qs_wait` から `qa_done`)。

## 第25章 ランダム関数タップセット

以下の関数は乱数の生成を行います。

### 名前

`function::randint` – 乱数  $[0,n)$  を返します。

### 概要

```
randint:long(n:long)
```

### 引数

***n***

範囲の上限を超える数字 ( $2^{**}20$  を超えない)。

## 第26章 文字列およびデータ取得関数タプセット

アドレスに基づいてカーネルまたはユーザー空間プログラムから文字列およびその他のプリミティブ型を取得する関数。すべての文字列の最大長は `MAXSTRINGLEN` で指定されます。

### 名前

`function::atomic_long_read` – カーネルメモリーからアトミック `long` 変数を取得します。

### 概要

```
atomic_long_read:long(addr:long)
```

### 引数

#### *addr*

アトミック `long` 変数のポインター。

### 説明

アトミック `long` 変数の読み取りを安全に実行します。これは、カーネル設定で `ATOMIC_LONG_INIT` が指定されていないカーネル上の `NOP` です。

---

### 名前

`function::atomic_read` – カーネルメモリーからアトミック変数を取得します。

### 概要

```
atomic_read:long(addr:long)
```

### 引数

#### *addr*

アトミック変数のポインター。

### 説明

アトミック変数の読み取りを安全に実行します。

---

### 名前

`function::kernel_char` – カーネルメモリーに保存された `char` 値を取得します。

### 概要

```
kernel_char:long(addr:long)
```

## 引数

### ***addr***

`char` 値の取得元のカーネルアドレス。

## 説明

指定のカーネルメモリーアドレスから `char` 値を返します。指定アドレスからの読み込みに失敗する場合にエラーを報告します。

---

## 名前

`function::kernel_int` – カーネルメモリーに保存される `int` 値を取得します。

## 概要

```
kernel_int:long(addr:long)
```

## 引数

### ***addr***

`int` の取得元のカーネルアドレスです。

## 説明

指定のカーネルメモリーアドレスから `int` 値を返します。指定アドレスからの読み込みに失敗する場合にエラーを報告します。

---

## 名前

`function::kernel_long` – カーネルメモリーに保存された `long` 値を取得します。

## 概要

```
kernel_long:long(addr:long)
```

## 引数

### ***addr***

`long` 値の取得元のカーネルアドレス。

## 説明

指定のカーネルメモリーアドレスから `long` 値を返します。指定アドレスからの読み込みに失敗する場合にエラーを報告します。

---

## 名前

**function::kernel\_pointer** – カーネルメモリーに保存されるポインター値を取得します。

## 概要

```
kernel_pointer:long(addr:long)
```

## 引数

### **addr**

ポインターの取得元のカーネルアドレス。

## 説明

指定のカーネルメモリーアドレスからポインター値を返します。指定アドレスからの読み込みに失敗する場合にエラーを報告します。

---

## 名前

**function::kernel\_short** – カーネルメモリーに保存される **short** 値を取得します。

## 概要

```
kernel_short:long(addr:long)
```

## 引数

### **addr**

**short** 値の取得元のカーネルアドレス。

## 説明

指定のカーネルメモリーアドレスから **short** 値を返します。指定アドレスからの読み込みに失敗する場合にエラーを報告します。

---

## 名前

**function::kernel\_string** – カーネルメモリーから文字列を取得します。

## 概要

```
kernel_string:string(addr:long)
```

## 引数

### **addr**

文字列の取得元のカーネルアドレス。

## 説明

この関数は、指定のカーネルメモリーアドレスから **NULL** 終端 **C** 文字列を返します。文字列のコピー障害のエラーを報告します。

---

## 名前

**function::kernel\_string2** – 代替エラー文字列と共にカーネルメモリーから文字列を取得します。

## 概要

```
kernel_string2:string(addr:long,err_msg:string)
```

## 引数

### **addr**

文字列の取得元のカーネルアドレス。

### **err\_msg**

データが利用できない場合に返すエラーメッセージ。

## 説明

この関数は、指定のカーネルメモリーアドレスから **NULL** 終端 **C** 文字列を返します。文字列のコピー障害の所定のエラーメッセージを報告します。

---

## 名前

**function::kernel\_string2\_utf16** – 代替エラー文字列と共にカーネルメモリーから **UTF-16** 文字列を取得します。

## 概要

```
kernel_string2_utf16:string(addr:long,err_msg:string)
```

## 引数

### **addr**

文字列の取得元のカーネルアドレス。

### **err\_msg**

データが利用できない場合に返すエラーメッセージ。

## 説明

この関数は、所定のカーネルメモリーアドレスの **UTF-16** 文字列から変換された **null** で終了した **UTF-8** 文字列を返します。文字列のコピー障害または変換エラーの所定のエラーメッセージを報告します。

---

## 名前

**function::kernel\_string2\_utf32** – 代替エラー文字列と共にカーネルメモリーから UTF-32 文字列を取得します。

## 概要

```
kernel_string2_utf32:string(addr:long,err_msg:string)
```

## 引数

### **addr**

文字列の取得元のカーネルアドレス。

### **err\_msg**

データが利用できない場合に返すエラーメッセージ。

## 説明

この関数は、所定のカーネルメモリーアドレスの UTF-32 文字列から変換された null で終了した UTF-8 文字列を返します。文字列のコピー障害または変換エラーの所定のエラーメッセージを報告します。

---

## 名前

**function::kernel\_string\_n** – カーネルメモリーから所定の長さの文字列を取得します。

## 概要

```
kernel_string_n:string(addr:long,n:long)
```

## 引数

### **addr**

文字列の取得元のカーネルアドレス。

### **n**

文字列の最大長 (null で終了していない場合)。

## 説明

所定のカーネルメモリーアドレスから最大長の C 文字列を返します。文字列のコピー障害のエラーを報告します。

---

## 名前

**function::kernel\_string\_quoted** – カーネルメモリーから文字列を取得し、引用符で囲みます。

## 概要

```
kernel_string_quoted:string(addr:long)
```

## 引数

### *addr*

文字列の取得元のカーネルメモリーアドレス。

## 説明

所定のカーネルメモリーアドレスから `null` で終了した C 文字列を返します。出力不可能な ASCII 文字は、返される文字列の対応するエスケープシーケンスで置き換えられます。文字列は二重引用符で囲まれることに注意してください。該当するアドレスでカーネルメモリーデータがアクセス不可能な場合は、アドレス自体が二重引用符なしで文字列として返されます。

---

## 名前

`function::kernel_string_quoted_utf16` – 該当するカーネル UTF-16 文字列を引用符で囲みます。

## 概要

```
kernel_string_quoted_utf16:string(addr:long)
```

## 引数

### *addr*

文字列の取得元のカーネルアドレス。

## 説明

この関数は、***string\_quoted*** に基づく引用符の追加と ***kernel\_string\_utf16*** に基づく UTF-16 デコーディングを組み合わせます。

---

## 名前

`function::kernel_string_quoted_utf32` – 該当する UTF-32 カーネル文字列を引用符で囲みます。

## 概要

```
kernel_string_quoted_utf32:string(addr:long)
```

## 引数

### *addr*

文字列の取得元のカーネルアドレス。

## 説明



この関数は、***string\_quoted***に基づく引用符の追加と ***kernel\_string\_utf32***に基づく UTF-32 デコーディングを組み合わせます。

---

## 名前

**function::kernel\_string\_utf16** – カーネルメモリーから UTF-16 文字列を取得します。

## 概要

```
kernel_string_utf16:string(addr:long)
```

## 引数

### ***addr***

文字列の取得元のカーネルアドレス。

## 説明

この関数は、所定のカーネルメモリーアドレスの UTF-16 文字列から変換された null で終了した UTF-8 文字列を返します。文字列のコピー障害または変換エラーでエラーを報告します。

---

## 名前

**function::kernel\_string\_utf32** – カーネルメモリーから UTF-32 文字列を取得します。

## 概要

```
kernel_string_utf32:string(addr:long)
```

## 引数

### ***addr***

文字列の取得元のカーネルアドレス。

## 説明

この関数は、所定のカーネルメモリーアドレスの UTF-32 文字列から変換された null で終了した UTF-8 文字列を返します。文字列のコピー障害または変換エラーでエラーを報告します。

---

## 名前

**function::user\_char** – ユーザー空間に保存された char 値を取得します。

## 概要

```
user_char:long(addr:long)
```

## 引数

### **addr**

char の取得元のユーザー空間アドレス。

## 説明

所定のユーザー空間アドレスから **char** 値を返します。ユーザー空間データにアクセスできない場合はゼロを返します。

---

## 名前

**function::user\_char\_warn** – ユーザー空間に保存された **char** 値を取得します。

## 概要

```
user_char_warn:long(addr:long)
```

## 引数

### **addr**

char の取得元のユーザー空間アドレス。

## 説明

所定のユーザー空間アドレスから **char** 値を返します。ユーザー空間データにアクセスできない場合はゼロを返し、障害に関して警告します (ただし、中止しません)。

---

## 名前

**function::user\_int** – ユーザー空間に保存された **int** 値を取得します。

## 概要

```
user_int:long(addr:long)
```

## 引数

### **addr**

int 値の取得元のユーザー空間アドレス。

## 説明

指定のユーザー空間アドレスから **int** 値を返します。ユーザー空間データにアクセスできない場合は 0 を返します。

---

## 名前

**function::user\_int16** – ユーザー空間に保存された 16 ビット整数値を取得します。

## 概要

```
user_int16:long(addr:long)
```

## 引数

### **addr**

16 ビット整数値の取得元のユーザー空間アドレス。

## 説明

所定のユーザー空間アドレスから 16 ビット整数値を返します。ユーザー空間データにアクセスできない場合はゼロを返します。

---

## 名前

**function::user\_int32** – ユーザー空間に保存された 32 ビット整数値を取得します。

## 概要

```
user_int32:long(addr:long)
```

## 引数

### **addr**

32 ビット整数値の取得元のユーザー空間アドレス。

## 説明

所定のユーザー空間アドレスから 32 ビット整数値を返します。ユーザー空間データにアクセスできない場合はゼロを返します。

---

## 名前

**function::user\_int64** – ユーザー空間に保存された 64 ビット整数値を取得します。

## 概要

```
user_int64:long(addr:long)
```

## 引数

### **addr**

64 ビット整数値の取得元のユーザー空間アドレス。

## 説明

所定のユーザー空間アドレスから **64** ビット整数値を返します。ユーザー空間データにアクセスできない場合はゼロを返します。

---

## 名前

**function::user\_int8** – ユーザー空間に保存された **8** ビット整数値を取得します。

## 概要

```
user_int8:long(addr:long)
```

## 引数

### **addr**

**8** ビット整数値の取得元のユーザー空間アドレス。

## 説明

所定のユーザー空間アドレスから **8** ビット整数値を返します。ユーザー空間データにアクセスできない場合はゼロを返します。

---

## 名前

**function::user\_int\_warn** – ユーザー空間に保存された **int** 値を取得します。

## 概要

```
user_int_warn:long(addr:long)
```

## 引数

### **addr**

**int** 値の取得元のユーザー空間アドレス。

## 説明

所定のユーザー空間アドレスから **int** 値を返します。ユーザー空間データにアクセスできない場合はゼロを返し、障害に関して警告します (ただし、中止しません)。

---

## 名前

**function::user\_long** – ユーザー空間に保存された **long** 値を取得します。

## 概要

```
user_long:long(addr:long)
```

## 引数

### **addr**

long 値の取得元のユーザー空間アドレス。

## 説明

所定のユーザー空間アドレスから long 値を返します。ユーザー空間データにアクセスできない場合はゼロを返します。long 値のサイズは、現在のユーザー空間タスクのアーキテクチャーによって異なることに注意してください (64/32 ビット互換タスクの両方をサポートするアーキテクチャーの場合)。

---

## 名前

function::user\_long\_warn – ユーザー空間に保存された long 値を取得します。

## 概要

```
user_long_warn:long(addr:long)
```

## 引数

### **addr**

long 値の取得元のユーザー空間アドレス。

## 説明

所定のユーザー空間アドレスから long 値を返します。ユーザー空間データにアクセスできない場合はゼロを返します。long 値のサイズは、現在のユーザー空間タスクのアーキテクチャーによって異なることに注意してください (64/32 ビット互換タスクの両方をサポートするアーキテクチャーの場合)。

---

## 名前

function::user\_short – ユーザー空間に保存された short 値を取得します。

## 概要

```
user_short:long(addr:long)
```

## 引数

### **addr**

short 値の取得元のユーザー空間アドレスです。

## 説明

指定のユーザー空間アドレスから short 値を返します。ユーザー空間データにアクセスできない場合は 0 を返します。

---

## 名前

**function::user\_short\_warn** – ユーザー空間に保存された **short** 値を取得します。

## 概要

```
user_short_warn:long(addr:long)
```

## 引数

### **addr**

**short** 値の取得元のユーザー空間アドレスです。

## 説明

所定のユーザー空間アドレスから **short** 値を返します。ユーザー空間データにアクセスできない場合はゼロを返し、障害に関して警告します (ただし、中止しません)。

---

## 名前

**function::user\_string** – ユーザー空間から文字列を取得します。

## 概要

```
user_string:string(addr:long)
```

## 引数

### **addr**

文字列の取得元のユーザー空間アドレス

## 説明

所定のユーザー空間メモリーアドレスから **NULL** 終端 **C** 文字列を返します。ユーザー空間データにアクセスできない稀なケースでエラーを報告します。

---

## 名前

**function::user\_string2** – 代替エラー文字列と共にユーザー空間から文字列を取得します。

## 概要

```
user_string2:string(addr:long,err_msg:string)
```

## 引数

### **addr**

文字列の取得元のユーザー空間アドレス

**err\_msg**

データが利用できない場合に返すエラーメッセージ。

**説明**

所定のユーザー空間メモリーアドレスから **NULL** 終端 **C** 文字列を返します。ユーザー空間データにアクセスできない稀なケースで所定のエラーメッセージを報告します。

**名前**

**function::user\_string2\_n\_warn** – 代替警告文字列と共にユーザー空間から文字列を取得します。

**概要**

```
user_string2_n_warn:string(addr:long,n:long,warn_msg:string)
```

**引数****addr**

文字列の取得元のユーザー空間アドレス

**n**

文字列の最大長 (**null** で終了していない場合)。

**warn\_msg**

データが利用できない場合に返す警告メッセージ。

**説明**

所定のユーザー空間メモリーアドレスから **C** 文字列の **n** 文字までを返します。ユーザー空間データにアクセスできない場合 (稀なケース) は所定の警告メッセージを報告し、障害に関して警告します (ただし、中止しません)。

**名前**

**function::user\_string2\_utf16** – 代替エラー文字列と共にユーザーメモリーから **UTF-16** 文字列を取得します。

**概要**

```
user_string2_utf16:string(addr:long,err_msg:string)
```

**引数****addr**

文字列の取得元のユーザーアドレス。

**err\_msg**

データが利用できない場合に返すエラーメッセージ。

## 説明

この関数は、所定のユーザーメモリーアドレスの UTF-16 文字列から変換された null で終了した UTF-8 文字列を返します。文字列のコピー障害または変換エラーの所定のエラーメッセージを報告します。

---

## 名前

**function::user\_string2\_utf32** – 代替エラー文字列と共にユーザーメモリーから UTF-32 文字列を取得します。

## 概要

```
user_string2_utf32:string(addr:long,err_msg:string)
```

## 引数

### **addr**

文字列の取得元のユーザーアドレス。

### **err\_msg**

データが利用できない場合に返すエラーメッセージ。

## 説明

この関数は、所定のユーザーメモリーアドレスの UTF-32 文字列から変換された null で終了した UTF-8 文字列を返します。文字列のコピー障害または変換エラーの所定のエラーメッセージを報告します。

---

## 名前

**function::user\_string2\_warn** – 代替警告文字列と共にユーザー空間から文字列を取得します。

## 概要

```
user_string2_warn:string(addr:long,warn_msg:string)
```

## 引数

### **addr**

文字列の取得元のユーザー空間アドレス

### **warn\_msg**

データが利用できない場合に返す警告メッセージ。

## 説明



所定のユーザー空間メモリアドレスから **NULL** 終端 **C** 文字列を返します。ユーザー空間データにアクセスできない場合 (稀なケース) は所定の警告メッセージを報告し、障害に関して警告します (ただし、中止しません)。

---

## 名前

**function::user\_string\_n** – ユーザー空間から所定の長さの文字列を取得します。

## 概要

```
user_string_n:string(addr:long,n:long)
```

## 引数

### **addr**

文字列の取得元のユーザー空間アドレス

### **n**

文字列の最大長 (**null** で終了していない場合)。

## 説明

所定のユーザー空間アドレスから最大長の **C** 文字列を返します。所定アドレスのユーザー空間データにアクセスできない稀なケースでエラーを返します。

---

## 名前

**function::user\_string\_n2** – ユーザー空間から所定の長さの文字列を取得します。

## 概要

```
user_string_n2:string(addr:long,n:long,err_msg:string)
```

## 引数

### **addr**

文字列の取得元のユーザー空間アドレス

### **n**

文字列の最大長 (**null** で終了していない場合)。

### **err\_msg**

データが利用できない場合に返すエラーメッセージ。

## 説明

所定のユーザー空間アドレスから最大長の **C** 文字列を返します。所定アドレスのユーザー空間データにアクセスできない稀なケースで所定のエラーメッセージ文字列を返します。

---

## 名前

**function::user\_string\_n2\_quoted** – ユーザー空間から文字列を取得し、引用符で囲みます。

## 概要

```
user_string_n2_quoted:string(addr:long,inlen:long,outlen:long)
```

## 引数

### **addr**

文字列の取得元のユーザー空間アドレス

### **inlen**

読み取る文字列の最大長 (**null** で終了していない場合)。

### **outlen**

出力文字列の最大長。

## 説明

所定のユーザー空間メモリーアドレスから C 文字列の **inlen** 文字までを読み取り、**outlen** 文字までを返します。出力不可能な **ASCII** 文字は、返される文字列の対応するエスケープシーケンスで置き換えられます。文字列は二重引用符で囲まれることに注意してください。該当するアドレスでユーザー空間データがアクセス不可能な場合は、アドレス自体が二重引用符なしで文字列として返されます。

---

## 名前

**function::user\_string\_n\_quoted** – ユーザー空間から文字列を取得し、引用符で囲みます。

## 概要

```
user_string_n_quoted:string(addr:long,n:long)
```

## 引数

### **addr**

文字列の取得元のユーザー空間アドレス

### **n**

文字列の最大長 (**null** で終了していない場合)。

## 説明

所定のユーザー空間メモリーアドレスから C 文字列の **n** 文字までを返します。出力不可能な **ASCII** 文字は、返される文字列の対応するエスケープシーケンスで置き換えられます。文字列は二重引用符で囲まれることに注意してください。該当するアドレスでユーザー空間データがアクセス不可能な場合は、アドレス自体が二重引用符なしで文字列として返されます。

## 名前

**function::user\_string\_n\_warn** – ユーザー空間から文字列を取得します。

## 概要

```
user_string_n_warn:string(addr:long,n:long)
```

## 引数

### **addr**

文字列の取得元のユーザー空間アドレス

### **n**

文字列の最大長 (null で終了していない場合)。

## 説明

所定のユーザー空間メモリーアドレスから **n** 文字までの **C** 文字列を返します。ユーザー空間データにアクセスできない稀な場合は「<unknown>」を報告し、障害について警告します (中止しません)。

## 名前

**function::user\_string\_quoted** – ユーザー空間から文字列を取得し、引用符で囲みます。

## 概要

```
user_string_quoted:string(addr:long)
```

## 引数

### **addr**

文字列の取得元のユーザー空間アドレス

## 説明

所定のユーザー空間メモリーアドレスから **NULL** 終端 **C** 文字列を返します。出力不可能な **ASCII** 文字は、返される文字列の対応するエスケープシーケンスで置き換えられます。文字列は二重引用符で囲まれることに注意してください。該当するアドレスでユーザー空間データがアクセス不可能な場合は、アドレス自体が二重引用符なしで文字列として返されます。

## 名前

**function::user\_string\_quoted\_utf16** – 該当するユーザー **UTF-16** 文字列を引用符で囲みます。

## 概要

```
user_string_quoted_utf16:string(addr:long)
```

## 引数

### **addr**

文字列の取得元のユーザーアドレス。

## 説明

この関数は、**string\_quoted**に基づく引用符の追加と **user\_string\_utf16**に基づく UTF-16 デコーディングを組み合わせます。

---

## 名前

function::user\_string\_quoted\_utf32 – 該当するユーザー UTF-32 文字列を引用符で囲みます。

## 概要

```
user_string_quoted_utf32:string(addr:long)
```

## 引数

### **addr**

文字列の取得元のユーザーアドレス。

## 説明

この関数は、**string\_quoted**に基づく引用符の追加と **user\_string\_utf32**に基づく UTF-32 デコーディングを組み合わせます。

---

## 名前

function::user\_string\_utf16 – ユーザーメモリーから UTF-16 文字列を取得します。

## 概要

```
user_string_utf16:string(addr:long)
```

## 引数

### **addr**

文字列の取得元のユーザーアドレス。

## 説明

この関数は、所定のユーザーメモリーアドレスの UTF-16 文字列から変換された null で終了した UTF-8 文字列を返します。文字列のコピー障害または変換エラーでエラーを報告します。

---

## 名前

**function::user\_string\_utf32** – ユーザーメモリーから UTF-32 文字列を取得します。

## 概要

```
user_string_utf32:string(addr:long)
```

## 引数

### **addr**

文字列の取得元のユーザーアドレス。

## 説明

この関数は、所定のユーザーメモリーアドレスの UTF-32 文字列から変換された null で終了した UTF-8 文字列を返します。文字列のコピー障害または変換エラーでエラーを報告します。

---

## 名前

**function::user\_string\_warn** – ユーザー空間から文字列を取得します。

## 概要

```
user_string_warn:string(addr:long)
```

## 引数

### **addr**

文字列の取得元のユーザー空間アドレス

## 説明

指定のユーザー空間メモリーアドレスから NULL 終端 C 文字列を返します。ユーザー空間データにアクセスできない場合に "" を報告し、障害ついて警告します (中止しません)。

---

## 名前

**function::user\_uint16** – ユーザー空間に保存された符号なし 16 ビット整数値を取得します。

## 概要

```
user_uint16:long(addr:long)
```

## 引数

### **addr**

符号なし 16 ビット整数値の取得元のユーザー空間アドレス。

## 説明

所定のユーザー空間アドレスから符号なし 16 ビット整数値を返します。ユーザー空間データにアクセスできない場合はゼロを返します。

---

## 名前

**function::user\_uint32** – ユーザー空間に保存された符号なし 32 ビット整数値を取得します。

## 概要

```
user_uint32:long(addr:long)
```

## 引数

### *addr*

符号なし 32 ビット整数値の取得元のユーザー空間アドレス。

## 説明

所定のユーザー空間アドレスから符号なし 32 ビット整数値を返します。ユーザー空間データにアクセスできない場合はゼロを返します。

---

## 名前

**function::user\_uint64** – ユーザー空間に保存された符号なし 64 ビット整数値を取得します。

## 概要

```
user_uint64:long(addr:long)
```

## 引数

### *addr*

符号なし 64 ビット整数値の取得元のユーザー空間アドレス。

## 説明

所定のユーザー空間アドレスから符号なし 64 ビット整数値を返します。ユーザー空間データにアクセスできない場合はゼロを返します。

---

## 名前

**function::user\_uint8** – ユーザー空間に保存された符号なし 8 ビット整数値を取得します。

## 概要

```
user_uint8:long(addr:long)
```

## 引数

### **addr**

符号なし 8 ビット整数値の取得元のユーザー空間アドレス。

## 説明

所定のユーザー空間アドレスから符号なし 8 ビット整数値を返します。ユーザー空間データにアクセスできない場合はゼロを返します。

---

## 名前

**function::user\_ulong** – ユーザー空間に保存された符号なし long 値を取得します。

## 概要

```
user_ulong:long(addr:long)
```

## 引数

### **addr**

符号なし long 値の取得元のユーザー空間アドレスです。

## 説明

指定のユーザー空間アドレスから符号なし long 値を返します。ユーザー空間データにアクセスできない場合は 0 を返します。符号なし long 値のサイズは、現在のユーザー空間タスクのアーキテクチャーによって異なることに注意してください (64/32 ビット互換タスクをサポートするアーキテクチャーの場合)。

---

## 名前

**function::user\_ulong\_warn** – ユーザー空間に保存された符号なし long 値を取得します。

## 概要

```
user_ulong_warn:long(addr:long)
```

## 引数

### **addr**

符号なし long 値の取得元のユーザー空間アドレスです。

## 説明

指定のユーザー空間アドレスから符号なし long 値を返します。ユーザー空間データにアクセスできない場合は 0 を返し、障害についての警告が出されます (中止はしない)。符号なし long 値のサイズは、現在のユーザー空間タスクのアーキテクチャーによって異なることに注意してください (64/32 ビット互換タスクをサポートするアーキテクチャーの場合)。

## 名前

**function::user\_ushort** – ユーザー空間に保存された符号なし **short** 値を取得します。

## 概要

```
user_ushort:long(addr:long)
```

## 引数

### **addr**

符号なし **short** 値の取得元のユーザー空間アドレスです。

## 説明

所定のユーザー空間アドレスから符号なし **short** 値を返します。ユーザー空間データにアクセスできない場合はゼロを返します。

---

## 名前

**function::user\_ushort\_warn** – ユーザー空間に保存された符号なし **short** 値を取得します。

## 概要

```
user_ushort_warn:long(addr:long)
```

## 引数

### **addr**

符号なし **short** 値の取得元のユーザー空間アドレスです。

## 説明

指定のユーザー空間アドレスから符号なし **short** 値を返します。ユーザー空間データにアクセスできない場合は **0** を返し、障害についての警告が出されます (中止はしない)。



## 第27章 文字列およびデータ書き込み関数タップセット

**SystemTap guru** モードは、障害をシミュレーションすることによってカーネルコードのエラー処理をテストするために使用できます。このタップセットの関数は、カーネルメモリーのプリミティブタイプに書き込む標準的な方法を提供します。このタップセットのすべての関数は **guru** モード (**-g**) を使用する必要があります。

### 名前

**function::set\_kernel\_char** – カーネルメモリーに **char** 値を書き込みます。

### 概要

```
set_kernel_char(addr:long, val:long)
```

### 引数

#### **addr**

**char** 値を書き込むカーネルアドレス。

#### **val**

書き込む **char**。

### 説明

指定のカーネルメモリーアドレスに **char** 値を書き込みます。指定アドレスへの書き込みが失敗した場合にエラーを報告します。**guru** モード (**-g**) を使用する必要があります。

---

### 名前

**function::set\_kernel\_int** – カーネルメモリーに **int** 値を書き込みます。

### 概要

```
set_kernel_int(addr:long, val:long)
```

### 引数

#### **addr**

**int** 値を書き込むカーネルアドレス。

#### **val**

書き込む **int** 値。

### 説明

指定のカーネルメモリーアドレスに **int** 値を書き込みます。指定アドレスへの書き込みが失敗した場合にエラーを報告します。**guru** モード (**-g**) を使用する必要があります。

---

## 名前

**function::set\_kernel\_long** – カーネルメモリーに **long** 値を書き込みます。

## 概要

```
set_kernel_long(addr:long, val:long)
```

## 引数

### **addr**

**long** 値を書き込むカーネルアドレス。

### **val**

書き込む **long** 値。

## 説明

指定のカーネルメモリーアドレスに **long** 値を書き込みます。指定アドレスへの書き込みが失敗した場合にエラーを報告します。**guru** モード (**-g**) を使用する必要があります。

---

## 名前

**function::set\_kernel\_pointer** – カーネルメモリーにポインター値を書き込みます。

## 概要

```
set_kernel_pointer(addr:long, val:long)
```

## 引数

### **addr**

ポインター値を書き込むカーネルアドレス。

### **val**

書き込むポインター値。

## 説明

指定のカーネルメモリーアドレスにポインター値を書き込みます。指定アドレスへの書き込みが失敗した場合にエラーを報告します。**guru** モード (**-g**) を使用する必要があります。

---

## 名前

**function::set\_kernel\_short** – カーネルメモリーに **short** 値を書き込みます。

## 概要

```
set_kernel_short(addr:long, val:long)
```

## 引数

### **addr**

short 値を書き込むカーネルアドレス。

### **val**

書き込む short 値。

## 説明

指定のカーネルメモリーアドレスに **short** 値を書き込みます。指定アドレスへの書き込みが失敗した場合にエラーを報告します。**guru** モード (**-g**) を使用する必要があります。

## 名前

**function::set\_kernel\_string** – カーネルメモリーに文字列を書き込みます。

## 概要

```
set_kernel_string(addr:long, val:string)
```

## 引数

### **addr**

文字列を書き込むカーネルアドレス。

### **val**

書き込む文字列。

## 説明

指定のカーネルメモリーアドレスに指定の文字列を書き込みます。文字列のコピーに失敗した場合にエラーを報告します。**guru** モード (**-g**) を使用する必要があります。

## 名前

**function::set\_kernel\_string\_n** – カーネルメモリーに所定の長さの文字列を書き込みます。

## 概要

```
set_kernel_string_n(addr:long, n:long, val:string)
```

## 引数

### **addr**

文字列を書き込むカーネルアドレス。

***n***

文字列の最大長。

***val***

書き込む文字列。

## 説明

指定のカーネルメモリーアドレスに最大長までの指定の文字列を書き込みます。文字列のコピーに失敗した場合にエラーを報告します。**guru** モード (**-g**) を使用する必要があります。

## 第28章 GURU タップセット

障害を挿入したり、可観測性を改善したりするためにシステムの動作に意図的に干渉する関数。このタップセットのすべての関数は **guru** モード (**-g**) を使用する必要があります。

### 名前

**function::mdelay** – ミリ秒単位の遅延。

### 概要

```
mdelay(ms:long)
```

### 引数

**ms**

遅延時間 (ミリ秒単位)。

### 説明

この関数は、ビジー状態の遅延時間 (ミリ秒単位) をプローブハンドラーに挿入します。**guru** モードを使用する必要があります。

---

### 名前

**function::panic** – パニックをトリガーします。

### 概要

```
panic(msg:string)
```

### 引数

**msg**

カーネルの **panic** 関数に渡すメッセージ

### 説明

この関数は、実行中のカーネルの即時パニックをユーザー指定パニックメッセージとともにトリガーします。**guru** モードを使用する必要があります。

---

### 名前

**function::raise** – 現在のスレッドでシグナルを発生させます。

### 概要

```
raise(signo:long)
```

## 引数

### ***signo***

シグナル番号

## 説明

この関数は、現在のスレッドのカーネル **send\_sig** ルーチンを該当する生の未チェックシグナル番号で呼び出します。**send\_sig** が失敗した場合は、エラーが発生することがあります。**guru** モードを使用する必要があります。

---

## 名前

**function::udelay** – マイクロ秒単位の遅延。

## 概要

```
udelay(us:long)
```

## 引数

### ***us***

遅延時間 (マイクロ秒単位)。

## 説明

この関数は、ビジー状態の遅延時間 (マイクロ秒単位) をプローブハンドラーに挿入します。**guru** モードを使用する必要があります。

## 第29章 標準的な文字列関数のコレクション

長さ、サブ文字列の取得、個別の文字の取得、文字列の検索、エスケープ、トークン化および文字列の `long` への変換を実行する関数です。

### 名前

**function::isdigit** – 数字があるかどうかをチェックします。

### 概要

```
isdigit:long(str:string)
```

### 引数

**str**

チェックする文字列。

### 説明

文字列の最初の文字に数字 (0 から 9) があるかどうかをチェックします。 **true** の場合はゼロ以外の値を返し、 **false** の場合はゼロを返します。

---

### 名前

**function::isinstr** – 文字列が別の文字列のサブ文字列かどうかを返します。

### 概要

```
isinstr:long(s1:string, s2:string)
```

### 引数

**s1**

検索する文字列。

**s2**

検索するサブ文字列。

### 説明

この関数は、文字列 **s1** に **s2** が含まれる場合に 1 を返します。そうでない場合はゼロを返します。

---

### 名前

**function::str\_replace** – `str_replace` は、サブ文字列のすべてのインスタンスを別のものに置き換えます。

## 概要

```
str_replace:string(prnt_str:string, srch_str:string, rplc_str:string)
```

## 引数

### *prnt\_str*

検索し、置換する文字列。

### *srch\_str*

*prnt\_str* 文字列で検索するために使用されるサブ文字列。

### *rplc\_str*

*srch\_str* を置換するために使用されるサブ文字列。

## 説明

この関数は、サブ文字列が置換された所定の文字列を返します。

---

## 名前

`function::string_quoted` – 所定の文字列を引用符で囲みます。

## 概要

```
string_quoted:string(str:string)
```

## 引数

### *str*

文字列の取得元のカーネルアドレス。

## 説明

該当する文字列の引用符で囲まれた文字列バージョンを文字とともに返します。出力不可能な **ASCII** 文字は、返される文字列の対応するエスケープシーケンスで置き換えられます。文字列は二重引用符で囲まれることに注意してください。

---

## 名前

`function::stringat` – 文字列の所定位置の文字を返します。

## 概要

```
stringat:long(str:string, pos:long)
```

## 引数



**str**

文字の取得元の文字列。

**pos**

文字の取得元の位置 (最初の文字は 0)。

**説明**

この関数は文字列の所定の位置にある文字を返します。文字列に多くの文字が含まれない場合はゼロを返します。位置が範囲外である場合はエラーを報告します。

---

**名前**

**function::strlen** – 文字列の長さを返します。

**概要**

```
strlen:long(s:string)
```

**引数****s**

文字列。

**説明**

この関数は、ゼロから **MAXSTRINGLEN** までに設定できる文字列の長さを返します。

---

**名前**

**function::strtol** – **strtol** - 文字列を **long** に変換します。

**概要**

```
strtol:long(str:string, base:long)
```

**引数****str**

変換する文字列。

**base**

使用するベース。

**説明**

この関数は、数字の文字列表示を整数に変換します。**base** パラメーターは、文字列に想定される基数を示します (例: 16 進数の 16、8 進数の 8、2 進数の 2)。

---

## 名前

**function::substr** – サブ文字列を返します。

## 概要

```
substr:string(str:string, start:long, length:long)
```

## 引数

### **str**

サブ文字列の取得元の文字列。

### **start**

抽出される文字列の開始位置 (最初の文字は 0)。

### **length**

返す文字列の長さ。

## 説明

該当する長さの開始位置にある文字列のサブ文字列を返します (元の文字列の長さが **start + length** 未満である場合、または **length** が **MAXSTRINGLEN** より大きい場合はより小さい値)。

---

## 名前

**function::text\_str** – 文字列の出力できない文字をエスケープします。

## 概要

```
text_str:string(input:string)
```

## 引数

### **input**

エスケープする文字列。

## 説明

この関数は文字列引数を受け入れ、出力不可能なすべての **ASCII** 文字は、返される文字列の対応するエスケープシーケンスに置き換えられます。

---

## 名前

**function::text\_strn** – 文字列の出力できない文字をエスケープします。

## 概要

```
text_strn:string(input:string, len:long, quoted:long)
```

## 引数

### *input*

エスケープする文字列。

### *len*

返す文字列の最大長 (0 は MAXSTRINGLEN を意味します)。

### *quoted*

文字列を二重引用符で囲みます。入力文字列が切り捨てられる場合、2 つ目の引用符の後に「...」が続きます。

## 説明

この関数は指定された長さの文字列を受け入れ、出力不可能なすべての ASCII 文字は、返される文字列の対応するエスケープシーケンスに置き換えられます。

## 名前

`function::tokenize` – 文字列の次の空でないトークンを返します。

## 概要

```
tokenize:string(input:string, delim:string)
```

## 引数

### *input*

トークン化する文字列。空の場合、直前の `tokenize` の呼び出しで渡された文字列の空でないトークンを返します。

### *delim*

トークンを区切る文字セット。

## 説明

この関数は、トークンが `delim` 文字列の文字で区切られる所定の入力文字列にある空でないトークンを返します。入力文字列が空以外である場合、最初のトークンを返します。入力文字列が空の場合、直前の `tokenize` の呼び出しで渡される文字列にある次のトークンを返します。区切り文字がない場合は、残りの入力文字列全体が返されます。利用できるトークンがない場合は空が返されます。

## 第30章 ANSI 制御文字をログで使用するためのユーティリティー関数

ANSI 制御文字を使用してロギングするためのユーティリティー関数です。これを使用すると、カーソル位置およびログメッセージの文字色の出力および属性を操作できます。

### 名前

**function::ansi\_clear\_screen** – カーソルを左上に移動し、画面をクリアします。

### 概要

```
ansi_clear_screen()
```

### 引数

なし

### 説明

カーソルを左上に移動するために **ansi** コードを送信し、カーソル位置から終了位置までの画面をクリアするために **ansi** コードを送信します。

---

### 名前

**function::ansi\_cursor\_hide** – カーソルを非表示にします。

### 概要

```
ansi_cursor_hide()
```

### 引数

なし

### 説明

カーソルを非表示にするために **ansi** コードを送信します。

---

### 名前

**function::ansi\_cursor\_move** – カーソルを新規の座標に移動します。

### 概要

```
ansi_cursor_move(x:long,y:long)
```

### 引数

**x**

カーソルの移動先の行。

**y**

カーソルの移動先の列。

## 説明

カーソルを **x** 行と **y** 列に配置するための ANSI コードを送信します。座標は 1 から始まり、(1,1) は左上を指します。

---

## 名前

**function::ansi\_cursor\_restore** – 以前に保存されたカーソル位置を復元します。

## 概要

```
ansi_cursor_restore()
```

## 引数

なし

## 説明

**ansi\_cursor\_save** で以前に保存した現在のカーソル位置を復元するために **ansi** コードを送信します。

---

## 名前

**function::ansi\_cursor\_save** – カーソル位置を保存します。

## 概要

```
ansi_cursor_save()
```

## 引数

なし

## 説明

現在のカーソル位置を保存するために **ansi** コードを送信します。

---

## 名前

**function::ansi\_cursor\_show** – カーソルを表示します。

## 概要

```
ansi_cursor_show()
```

## 引数

なし

## 説明

カーソルを表示するために **ansi** コードを送信します。

---

## 名前

**function::ansi\_new\_line** – カーソルを新しい行に移動します。

## 概要

```
ansi_new_line()
```

## 引数

なし

## 説明

改行の **ANSI** コードを送信します。

---

## 名前

**function::ansi\_reset\_color** – Select Graphic Rendition モードをリセットします。

## 概要

```
ansi_reset_color()
```

## 引数

なし

## 説明

前景色、背景色および色属性をデフォルト値にリセットするために **ansi** コードを送信します。

---

## 名前

**function::ansi\_set\_color** – ansi Select Graphic Rendition モードを設定します。

## 概要

```
ansi_set_color(fg:long)
```

## 引数

**fg**

設定する前景色。

## 説明

前景色を指定する Select Graphic Rendition モードの ANSI コードを送信します。黒 (30)、青 (34)、緑 (32)、シアン (36)、赤 (31)、紫 (35)、茶 (33)、灰 (37)。

---

## 名前

function::ansi\_set\_color2 – ansi Select Graphic Rendition モードを設定します。

## 概要

```
ansi_set_color2(fg:long,bg:long)
```

## 引数

### *fg*

設定する前景色。

### *bg*

設定する背景色。

## 説明

前景色 (黒 (30)、青 (34)、緑 (32)、シアン (36)、赤 (31)、紫 (35)、茶 (33)、灰 (37)) および背景色 (黒 (30)、赤 (41)、緑 (42)、黄 (43)、青 (44)、紫 (45)、シアン (36)、白 (47)) を指定する Select Graphic Rendition モードの ANSI コードを送信します。

---

## 名前

function::ansi\_set\_color3 – ansi Select Graphic Rendition モードを設定します。

## 概要

```
ansi_set_color3(fg:long,bg:long,attr:long)
```

## 引数

### *fg*

設定する前景色。

### *bg*

設定する背景色。

### *attr*

設定する色属性。

## 説明

前景色 (黒 (30)、青 (34)、緑 (32)、シアン (36)、赤 (31)、紫 (35)、茶 (33)、灰 (37))、および背景色 (黒 (40)、赤 (41)、緑 (42)、黄 (43)、青 (44)、紫 (45)、シアン (46)、白 (47))、およびすべての属性を解

除 (0)、太字 (1)、下線 (4)、点滅 (5)、高速点滅 (6)、反転 (7)などの色属性を指定する **Select Graphic Rendition** モードの ANSI コードを送信します。

---

## 名前

**function::indent** – インデントするスペースの量を返します。

## 概要

```
indent:string(delta:long)
```

## 引数

### *delta*

各呼び出しに対して追加または削除されたスペースの量。

## 説明

この関数は、適切にインデントされた文字列を返します。小さい正または一致する負のデルタで呼び出します。**thread\_indent** 関数とは異なり、インデントではスレッドごとに個別のインデント値を追跡しません。

---

## 名前

**function::indent\_depth** – グローバルなネストの深さを返します。

## 概要

```
indent_depth:long(delta:long)
```

## 引数

### *delta*

各呼び出しに対して追加または削除された深さ。

## 説明

この関数は、適切なインデントのために数字を返します (**indent** に似ています)。小さい正または一致する負のデルタで呼び出します。**thread\_indent\_depth** 関数とは異なり、インデントではスレッドごとに個別のインデント値を追跡しません。

---

## 名前

**function::thread\_indent** – 現在のタスク情報とともにスペースの量を返します。

## 概要

```
thread_indent:string(delta:long)
```



## 引数

### *delta*

各呼び出しに対して追加または削除されたスペースの量。

## 説明

この関数は、スレッドのために適切にインデントされた文字列を返します。小さい正または一致する負のデルタで呼び出します。これが実際の最も外側の初期レベルのインデントの場合は、相対的なタイムスタンプベースがゼロにリセットされます。タイムスタンプは `__indent_timestamp` 関数 (デフォルトではマイクロ秒を測定します) により提供されます。

## 名前

`function::thread_indent_depth` – 現在のタスクのネストの深さを返します。

## 概要

```
thread_indent_depth:long(delta:long)
```

## 引数

### *delta*

各呼び出しに対して追加または削除された深さ。

## 説明

この関数は、一番外側の初期レベルから始まる、ネストされた関数呼び出しの深さと等しい整数値を返します。この関数は、長いネストの関数呼び出しでトレースのスペース (ホワイトスペースにより消費されます) を節約する場合に役に立ちます。この関数を `thread_indent` と同様の方法で使用します。つまり、`call-probe` で、`thread_indent_depth(1)` を使用し、`return-probe` で `thread_indent_depth(-1)` を使用します。

## 第31章 SYSTEMTAP トランスレータータップセット

この種類のユーザー空間プローブポイントは、SystemTap トランスレーター (**stap**) の操作をプローブし、コマンド (**staprun**) を実行するために使用されます。このタップセットには、さまざまなフェーズの SystemTap とインストールメンテーションキャッシュの SystemTap の管理を監視するプローブが含まれます。以下のプローブポイントが含まれます。

### 名前

**probe::stap.cache\_add\_mod** – カーネルインストールメンテーションモジュールをキャッシュに追加します。

### 概要

```
stap.cache_add_mod
```

### 値

#### **dest\_path**

.ko ファイルの移動先パス (ファイル名を含む)。

#### **source\_path**

.ko ファイルの移動元パス (ファイル名を含む)。

### 説明

ファイルが実際に移動される直前に実行されます。移動に失敗した場合、**cache\_add\_src** と **cache\_add\_nss** は実行されません。

---

### 名前

**probe::stap.cache\_add\_nss** – NSS (Network Security Services) 情報をキャッシュに追加します。

### 概要

```
stap.cache_add_nss
```

### 値

#### **source\_path**

.sgn ファイルの移動元パス (ファイル名を含む)。

#### **dest\_path**

.sgn ファイルの移動元パス (ファイル名を含む)。

### 説明

ファイルが実際に移動される直前に実行されます。**stap** は NSS サポートとともにコンパイルする必要があります。カーネルモジュールの移動に失敗した場合、このプローブは実行されません。

---

## 名前

`probe::stap.cache_add_src` – C コード変換をキャッシュに追加します。

## 概要

`stap.cache_add_src`

## 値

### *dest\_path*

.c ファイルの移動先パス (ファイル名を含む)。

### *source\_path*

.c ファイルの移動元パス (ファイル名を含む)。

## 説明

ファイルが実際に移動される直前に実行されます。カーネルモジュールの移動に失敗した場合、このプローブは実行されません。

---

## 名前

`probe::stap.cache_clean` – `stap` キャッシュからファイルを削除します。

## 概要

`stap.cache_clean`

## 値

### *path*

削除される `ko/.c` ファイルのパス。

## 説明

モジュール/ソースファイルをリンク解除する呼び出しの直前に実行されます。

---

## 名前

`probe::stap.cache_get` – `stap` キャッシュ内で見つかったアイテム。

## 概要

`stap.cache_get`

## 値

### ***module\_path***

.ko カーネルモジュールファイルのパス。

### ***source\_path***

.c ソースファイルのパス。

## 説明

`get_from_cache` が返される直前に実行されます (キャッシュ取得が正常に行われた場合)。

---

## 名前

`probe::stap.pass0 – stap pass0` (コマンドライン引数の解析) を開始します。

## 概要

```
stap.pass0
```

## 値

### ***session***

`systemtap_session` 変数。

## 説明

コマンドライン引数が解析された後に `pass0` が実行されます。

---

## 名前

`probe::stap.pass0.end – 完了した stap pass0` (コマンドライン引数の解析)。

## 概要

```
stap.pass0.end
```

## 値

### ***session***

`systemtap_session` 変数。

## 説明

`pass1` に対する `gettimeofday` 呼び出しの直前に `pass0.end` が実行されます。

---

## 名前

probe::stap.pass1.end – 完了した `stap pass1` (スクリプトの解析)。

## 概要

```
stap.pass1.end
```

## 値

### *session*

`systemtap_session` 変数。

## 説明

`cleanup` if `s.last_pass = 1` へのジャンプの直前に `pass1.end` が実行されます。

---

## 名前

probe::stap.pass1a – `stap pass1` (ユーザースクリプトの解析) の開始。

## 概要

```
stap.pass1a
```

## 値

### *session*

`systemtap_session` 変数。

## 説明

`gettimeofday` に対する呼び出しの直後に `pass1a` が実行されます (ユーザースクリプトが解析される前)。

---

## 名前

probe::stap.pass1b – `stap pass1` (ライブラリースクリプトの解析) の開始。

## 概要

```
stap.pass1b
```

## 値

### *session*

`systemtap_session` 変数。

## 説明

ライブラリスクリプトが解析される直前に **pass1b** が実行されます。

---

## 名前

**probe::stap.pass2 – stap pass2** (エラボレーション) の開始。

## 概要

```
stap.pass2
```

## 値

### *session*

**systemtap\_session** 変数。

## 説明

**gettimeofday** に対する呼び出しの直後に **pass2** が実行されます (**semantic\_pass** に対する呼び出しの直前)。

---

## 名前

**probe::stap.pass2.end – 完了した stap pass2** (エラボレーション) 。

## 概要

```
stap.pass2.end
```

## 値

### *session*

**systemtap\_session** 変数。

## 説明

**cleanup** if **s.last\_pass = 2** へのジャンプの直前に **pass2.end** が実行されます。

---

## 名前

**probe::stap.pass3 – stap pass3** (C への変換) の開始。

## 概要

```
stap.pass3
```

## 値

**session**

systemtap\_session 変数。

**説明**

**gettimeofday** に対する呼び出しの直後に **pass3** が実行されます (**translate\_pass** に対する呼び出しの直前)。

---

**名前**

probe::stap.pass3.end – 完了した stap pass3 (C への変換)。

**概要**

```
stap.pass3.end
```

**値****session**

systemtap\_session 変数。

**説明**

cleanup if s.last\_pass = 3 へのジャンプの直前に **pass3.end** が実行されます。

---

**名前**

probe::stap.pass4 – stap pass4 (カーネルモジュールへの C コードのコンパイル) の開始

**概要**

```
stap.pass4
```

**値****session**

systemtap\_session 変数。

**説明**

**gettimeofday** に対する呼び出しの直後に **pass4** が実行されます (**compile\_pass** に対する呼び出しの直前)。

---

**名前**

probe::stap.pass4.end – 完了した stap pass4 (カーネルモジュールへの C コードのコンパイル)。

## 概要

```
stap.pass4.end
```

## 値

### *session*

systemtap\_session 変数。

## 説明

cleanup if s.last\_pass = 4 へのジャンプの直前に **pass4.end** が実行されます。

---

## 名前

probe::stap.pass5 – stap pass5 (インストルメンテーションの実行) の開始。

## 概要

```
stap.pass5
```

## 値

### *session*

systemtap\_session 変数。

## 説明

**gettimeofday** に対する呼び出しの直後に **pass5** が実行されます (**run\_pass** に対する呼び出しの直前)。

---

## 名前

probe::stap.pass5.end – 完了した stap pass5 (インストルメンテーションの実行)。

## 概要

```
stap.pass5.end
```

## 値

### *session*

systemtap\_session 変数。

## 説明

クリーンアップラベルの直前に **pass5.end** が実行されます。

---



## 名前

probe::stap.pass6 – stap pass6 (クリーンアップ) の開始。

## 概要

```
stap.pass6
```

## 値

### **session**

systemtap\_session 変数。

## 説明

クリーンアップラベルの直後に **pass6** が実行されます (基本的には **pass5.end** と同じ)。

---

## 名前

probe::stap.pass6.end – 完了した stap pass6 (クリーンアップ)。

## 概要

```
stap.pass6.end
```

## 値

### **session**

systemtap\_session 変数。

## 説明

**main** が戻る直前に **pass6.end** が実行されます。

---

## 名前

probe::stap.system – stap からコマンドを開始します。

## 概要

```
stap.system
```

## 値

### **command**

posix\_spawn により実行されるコマンド文字列 (sh -c <str> など)。

## 説明

`stap_system` コマンドのエントリーで実行されます。

---

## 名前

`probe::stap.system.return` – `stap` からコマンドを終了。

## 概要

```
stap.system.return
```

## 値

### *ret*

生成されたプロセスで実行中の `waitpid` に関連付けられた戻りコード。ゼロ以外の値はエラーを示します。

## 説明

`waitpid` の後に `stap_system` 関数が返される直前に実行されます。

---

## 名前

`probe::stap.system.spawn` – `stap` の生成された新規プロセス。

## 概要

```
stap.system.spawn
```

## 値

### *ret*

`posix_spawn` からの戻り値。

### *pid*

生成されたプロセスの PID。

## 説明

`posix_spawn` の呼び出しの直後に実行されます。

---

## 名前

`probe::stapio.receive_control_message` – 制御メッセージを受信。

## 概要

```
stapio.receive_control_message
```

## 値

### ***len***

データブロブの長さ (バイト単位)

### ***data***

制御メッセージとして送信されるデータのバイナリーブロブへのポインター。

### ***type***

送信されるメッセージのタイプ。runtime/transport/transport\_msgs.h で定義されます。

## 説明

メッセージの受信直後およびその処理前に実行されます。

---

## 名前

probe::staprun.insert\_module – System Tap インストールメンテーションモジュールの挿入。

## 概要

```
staprun.insert_module
```

## 値

### ***path***

.ko カーネルモジュールへの完全パスが挿入されます。

## 説明

モジュールを挿入する呼び出しの直前に実行されます。

---

## 名前

probe::staprun.remove\_module – System Tap インストールメンテーションモジュールの削除。

## 概要

```
staprun.remove_module
```

## 値

### ***name***

削除される stap モジュール名 (.ko 拡張子なし)。

## 説明

モジュールを削除する呼び出しの直前に実行されます。

---

## 名前

`probe::staprun.send_control_message` – 制御メッセージの送信。

## 概要

`staprun.send_control_message`

## 値

### *type*

送信されるメッセージのタイプ。 `runtime/transport/transport_msgs.h` で定義されます。

### *data*

制御メッセージとして送信されるデータのバイナリーブロブへのポインター。

### *len*

データブロブの長さ (バイト単位)

## 説明

`send_request` 関数の開始時に実行されます。

## 第32章 ネットワークファイルストレージタッパセット

この種類のプローブポイントは、ネットワークファイルストレージの機能および操作をプローブするために使用されます。

### 名前

**function::nfsderror** – nfsd エラー番号を文字列に変換します。

### 概要

```
nfsderror:string(err:long)
```

### 引数

**err**

errnum

### 説明

この関数は、この関数に渡されるエラー番号の文字列を返します。

### 名前

**probe::nfs.aop.readpage** – ページを同期的に読み取る NFS クライアント。

### 概要

```
nfs.aop.readpage
```

### 値

**size**

この実行で読み取られるページ数。

**i\_flag**

ファイルフラグ。

**file**

ファイル引数。

**ino**

inode 番号。

**i\_size**

ファイルの長さ (バイト単位)。

**dev**

デバイス ID。

***rsize***

読み取りサイズ (バイト単位)。

***\_\_page***

ページのアドレス。

***sb\_flag***

スーパーブロックフラグ。

***page\_index***

マッピング内のオフセット。ページ ID を使用し、ID をページフレームに位置付けることができます。

## 説明

ページの読み取り。直前の非同期読み取り操作が失敗した場合にのみ実行されます。

---

## 名前

probe::nfs.aop.readpages – 複数ページを読み取る NFS クライアント。

## 概要

```
nfs.aop.readpages
```

## 値

***nr\_pages***

この実行で読み取りを試行したページ数。

***ino***

inode 番号。

***file***

filp 引数。

***size***

この実行で読み取りを試行したページ数。

***rsize***

読み取りサイズ (バイト単位)。

***dev***

デバイス ID。

***rpages***

読み取りサイズ (ページ単位)。

**説明**

先読み方法 (*readahead way*) の場合に実行されます。1 回に複数ページを読み取ります。

---

**名前**

`probe::nfs.aop.release_page` – ページをリリースする NFS クライアント。

**概要**

```
nfs.aop.release_page
```

**値*****size***

ページのリリース。

***ino***

inode 番号。

***dev***

デバイス ID。

***\_\_page***

ページのアドレス。

***page\_index***

マッピング内のオフセット。ページ ID を使用し、ID をページフレームに位置付けることができます。

**説明**

NFS でリリース操作を実行する際に実行されます。

---

**名前**

`probe::nfs.aop.set_page_dirty` – ページを *dirty* とマークする NFS クライアント。

**概要**

```
nfs.aop.set_page_dirty
```

**値**

*page*

**\_\_page**

ページのアドレス。

**page\_flag**

ページフラグ。

## 説明

このプローブは `generic __set_page_dirty_nobuffers` 関数に接続します。そのため、このプローブは、NFS クライアントのほかに他の多くのファイルシステムで実行されます。

---

## 名前

`probe::nfs.aop.write_begin` – データの書き込みを開始する NFS クライアント。

## 概要

`nfs.aop.write_begin`

## 値

**\_\_page**

ページのアドレス。

**page\_index**

マッピング内のオフセット。ページ ID を使用し、ID をページフレームに位置付けることができます。

**size**

書き込みバイト。

**to**

この書き込み操作の終了アドレス。

**ino**

inode 番号。

**offset**

この書き込み操作の開始アドレス。

**dev**

デバイス ID。

## 説明

NFS で書き込み操作が実行される際に生じます。書き込み用のページを準備し、ページに対応する要求を検索します。ページが存在し、それが別のファイルに属する場合は、ページへのコピーを試行する前にページが破棄されます。さらに、既存の破棄されたページから要求を見つけた場合にも同じ処理をし



ます。

---

## 名前

`probe::nfs.aop.write_end` – データの書き込みを完了する NFS クライアント。

## 概要

`nfs.aop.write_end`

## 値

### ***sb\_flag***

スーパーブロックフラグ。

### ***\_\_page***

ページのアドレス。

### ***page\_index***

マッピング内のオフセット。ページ ID を使用し、ID をページフレームに位置付けることができます。

### ***to***

この書き込み操作の終了アドレス。

### ***ino***

inode 番号。

### ***i\_flag***

ファイルフラグ。

### ***size***

書き込みバイト。

### ***dev***

デバイス ID。

### ***offset***

この書き込み操作の開始アドレス。

### ***i\_size***

ファイルの長さ (バイト単位)。

## 説明

NFS で書き込み操作を実行する際に実行されます (`prepare_write` 後の場合が多い)。

NFS ファイルのキャッシュされたページを更新します。さらに書き込みを行う可能性があります。

---

## 名前

`probe::nfs.aop.writepage` – マップされたページを NFS サーバーに書き込む NFS クライアント。

## 概要

`nfs.aop.writepage`

## 値

### ***wsiz***

書き込みサイズ。

### ***size***

この実行で書き込まれるページ数。

### ***i\_flag***

ファイルフラグ。

### ***for\_kupdate***

`writeback_control` のフラグです。これが `kupdate writeback` であるかどうかを示します。

### ***ino***

inode 番号。

### ***i\_size***

ファイルの長さ (バイト単位)。

### ***dev***

デバイス ID。

### ***for\_reclaim***

`writeback_control` のフラグです。これがページアロケータから呼び出されたかどうかを示します。

### ***\_\_page***

ページのアドレス。

### ***sb\_flag***

スーパーブロックフラグ。

### ***page\_index***

マッピング内のオフセット。ページ ID を使用し、ID をページフレームに位置付けることができます。

***i\_state***

inode 状態フラグ。

**説明**

wb の優先順位は、***for\_reclaim*** および ***for\_kupdate*** のフラグによって決定されます。

---

**名前**

probe::nfs.aop.writepages – 複数の dirty ページを NFS サーバーに書き込む NFS クライアント。

**概要**

nfs.aop.writepages

**値*****for\_reclaim***

writeback\_control のフラグです。これがページアロケータから呼び出されたかどうかを示します。

***wpages***

書き込みサイズ (ページ単位)。

***nr\_to\_write***

この実行で書き込みを試行したページ数。

***for\_kupdate***

writeback\_control のフラグです。これが kupdate writeback であるかどうかを示します。

***ino***

inode 番号。

***size***

この実行で書き込みを試行したページ数。

***wsiz***

書き込みサイズ。

***dev***

デバイス ID。

**説明**

wb の優先順位は、***for\_reclaim*** および ***for\_kupdate*** のフラグによって決定されます。

---

## 名前

probe::nfs.fop.aio\_read – aio\_read ファイル操作を実行する NFS クライアント。

## 概要

nfs.fop.aio\_read

## 値

### **ino**

inode 番号。

### **cache\_time**

この inode の読み取りキャッシュの開始時。

### **file\_name**

ファイル名。

### **buf**

ユーザー空間のバッファアドレス。

### **dev**

デバイス ID。

### **pos**

ファイルの現在位置。

### **attrtimeo**

キャッシュされた情報が有効であると想定される期間。jiffies - read\_cache\_jiffies > attrtimeo の場合にこの inode のキャッシュされた属性を再検証する必要があります。

### **count**

読み取りサイズ。

### **parent\_name**

親ディレクトリ一名。

### **cache\_valid**

キャッシュ関連のビットマスクフラグ。

---

## 名前

probe::nfs.fop.aio\_write – aio\_write ファイル操作を実行する NFS クライアント。

## 概要

---

`nfs.fop.aio_write`

値

***count***

読み取りサイズ。

***parent\_name***

親ディレクトリー名。

***ino***

inode 番号。

***file\_name***

ファイル名。

***buf***

ユーザー空間のバッファアドレス。

***dev***

デバイス ID。

***pos***

ファイルのオフセット。

---

名前

`probe::nfs.fop.check_flags` – フラグ操作をチェックする NFS クライアント。

概要

`nfs.fop.check_flags`

値

***flag***

ファイルフラグ。

---

名前

`probe::nfs.fop.flush` – flush file 操作を実行する NFS クライアント。

概要

`nfs.fop.flush`

値

***ndirty***

dirty ページの数。

***ino***

inode 番号。

***mode***

ファイルモード。

***dev***

デバイス ID。

---

名前

`probe::nfs.fop.fsync` – `fsync` 操作を実行する NFS クライアント。

概要

`nfs.fop.fsync`

値

***ndirty***

dirty ページの数。

***ino***

inode 番号。

***dev***

デバイス ID。

---

名前

`probe::nfs.fop.llseek` – `llseek` 操作を実行する NFS クライアント。

概要

`nfs.fop.llseek`

値

**ino**

inode 番号。

**whence**

シーク元の位置。

**dev**

デバイス ID。

**offset**

ファイルのオフセットが再度位置付けられます。

**whence\_str**

シーク元の位置のシンボリック文字列表現。

---

## 名前

probe::nfs.fop.lock – file lock 操作を実行する NFS クライアント。

## 概要

nfs.fop.lock

## 値

**fl\_start**

ロックされたリージョンの始点オフセット。

**ino**

inode 番号。

**fl\_flag**

ロックフラグ。

**i\_mode**

ファイルタイプおよびアクセス権。

**dev**

デバイス ID。

**fl\_end**

ロックされたリージョンの終点オフセット。

**fl\_type**

ロックタイプ。

**cmd**

cmd 引数。

---

**名前**

probe::nfs.fop.mmap – mmap 操作を実行する NFS クライアント。

**概要**

**|** nfs.fop.mmap

**値****attrtimeo**

キャッシュされた情報が有効であると想定される期間。jiffies - read\_cache\_jiffies > attrtimeo の場合にこの inode のキャッシュされた属性を再検証する必要があります。

**vm\_end**

vm\_mm 内にある終了アドレスの後の最初のバイト。

**dev**

デバイス ID。

**buf**

ユーザー空間のバッファアドレス。

**vm\_flag**

vm フラグ。

**cache\_time**

この inode の読み取りキャッシュの開始時。

**file\_name**

ファイル名。

**ino**

inode 番号。

**cache\_valid**

キャッシュ関連のビットマスクフラグ。

**parent\_name**

親ディレクトリ一名。

**vm\_start**



vm\_mm 内の開始アドレス。

---

## 名前

probe::nfs.fop.open – file open 操作を実行する NFS クライアント。

## 概要

nfs.fop.open

## 値

### **flag**

ファイルフラグ。

### **i\_size**

ファイルの長さ (バイト単位)。

### **dev**

デバイス ID。

### **file\_name**

ファイル名。

### **ino**

inode 番号。

---

## 名前

probe::nfs.fop.read – read 操作を実行する NFS クライアント。

## 概要

nfs.fop.read

## 値

### **devname**

ブロックデバイス名

## 説明

SystemTap は `vfs.do_sync_read` プローブを使用してこのプローブを実装します。その結果、NFS クライアントの読み取り操作以外の操作を取得します。

---

## 名前

probe::nfs.fop.read\_iter – read\_iter file 操作を実行する NFS クライアント。

## 概要

`nfs.fop.read_iter`

## 値

### *ino*

inode 番号。

### *file\_name*

ファイル名。

### *cache\_time*

この inode の読み取りキャッシュの開始時。

### *pos*

ファイルの現在位置。

### *dev*

デバイス ID。

### *attrtimeo*

キャッシュされた情報が有効であると想定される期間。 `jiffies - read_cache_jiffies > attrtimeo` の場合にこの inode のキャッシュされた属性を再検証する必要があります。

### *count*

読み取りサイズ。

### *parent\_name*

親ディレクトリー名。

### *cache\_valid*

キャッシュ関連のビットマスクフラグ。

---

## 名前

probe::nfs.fop.release – release page 操作を実行する NFS クライアント。

## 概要

`nfs.fop.release`

## 値

### ***ino***

inode 番号。

### ***dev***

デバイス ID。

### ***mode***

ファイルモード。

---

## 名前

probe::nfs.fop.sendfile – send file 操作を実行する NFS クライアント。

## 概要

nfs.fop.sendfile

## 値

### ***cache\_valid***

キャッシュ関連のビットマスクフラグ。

### ***ppos***

ファイルの現在位置。

### ***count***

読み取りサイズ。

### ***dev***

デバイス ID。

### ***attrtimeo***

キャッシュされた情報が有効であると想定される期間。jiffies - read\_cache\_jiffies > attrtimeo の場合にこの inode のキャッシュされた属性を再検証する必要があります。

### ***ino***

inode 番号。

### ***cache\_time***

この inode の読み取りキャッシュの開始時。

---

## 名前

probe::nfs.fop.write – write 操作を実行する NFS クライアント。

## 概要

nfs.fop.write

## 値

### **devname**

ブロックデバイス名

## 説明

SystemTap は `vfs.do_sync_write` プローブを使用してこのプローブを実装します。その結果、NFS クライアントの書き込み操作以外の操作を取得します。

---

## 名前

probe::nfs.fop.write\_iter – write\_iter file 操作を実行する NFS クライアント。

## 概要

nfs.fop.write\_iter

## 値

### **parent\_name**

親ディレクトリー名。

### **count**

読み取りサイズ。

### **pos**

ファイルのオフセット。

### **dev**

デバイス ID。

### **file\_name**

ファイル名。

### **ino**

inode 番号。

---

## 名前

`probe::nfs.proc.commit` – サーバーでデータをコミットする NFS クライアント。

## 概要

`nfs.proc.commit`

## 値

### **size**

この実行での読み取りバイト。

### **prot**

転送プロトコル。

### **version**

NFS バージョン。

### **server\_ip**

サーバーの IP アドレス。

### **bitmask1**

このファイルシステムでサポートされる属性セットを表す V4 ビットマスク。

### **offset**

ファイルオフセット。

### **bitmask0**

このファイルシステムでサポートされる属性セットを表す V4 ビットマスク。

## 説明

2006 年 12 月にすべての `nfs.proc.commit kernel` 関数は `kernel commit 200baa` で削除されました。そのため、これらのプローブは Linux 2.6.21 以降のカーネルには存在しません。

クライアントがバッファされたデータをディスクに書き込む際に実行されます。バッファされたデータはそれ以前にクライアントによって非同期的に書き込まれます。`commit` 関数は同期的に機能します。このプローブポイントは NFSv2 には存在しません。

## 名前

`probe::nfs.proc.commit_done` – コミット RPC タスクに対応する NFS クライアント。

## 概要

`nfs.proc.commit_done`

## 値

### ***status***

最終操作の結果。

### ***server\_ip***

サーバーの IP アドレス。

### ***prot***

転送プロトコル。

### ***version***

NFS バージョン。

### ***count***

コミットされたバイト数。

### ***valid***

`fattr->valid` は有効なファイルを示します。

### ***timestamp***

リリースの更新に使用される V4 タイムスタンプ。

## 説明

コミット RPC タスクが受信されるか、一部のコミット操作のエラー (タイムアウトまたはソケットのシャットダウン) が発生する際に実行されます。

---

## 名前

`probe::nfs.proc.commit_setup` – コミット RPC タスクをセットアップする NFS クライアント。

## 概要

`nfs.proc.commit_setup`

## 値

### ***version***

NFS バージョン。

### ***count***

このコミットのバイト数。

### ***prot***

転送プロトコル。

**server\_ip**

サーバーの IP アドレス。

**bitmask1**

このファイルシステムでサポートされる属性セットを表す V4 ビットマスク。

**bitmask0**

このファイルシステムでサポートされる属性セットを表す V4 ビットマスク。

**offset**

ファイルオフセット。

**size**

このコミットのバイト数。

## 説明

`commit_setup` 関数はコミット RPC タスクをセットアップするために使用されます。これは実際のコミット操作を実行しません。NFSv2 には存在しません。

---

## 名前

`probe::nfs.proc.create` – サーバー上でファイルを作成する NFS クライアント。

## 概要

`nfs.proc.create`

## 値

**server\_ip**

サーバーの IP アドレス。

**prot**

転送プロトコル。

**version**

NFS バージョン (この関数はすべての NFS バージョンに使用されます)。

**filename**

ファイル名。

**fh**

親ディレクトリーのファイルハンドル。

**filelen**

ファイル名の長さ。

**flag**

create モードであることを示します (NFSv3 および NFSv4 の場合のみ)。

---

## 名前

probe::nfs.proc.handle\_exception – NFSv4 例外を処理する NFS クライアント。

## 概要

```
nfs.proc.handle_exception
```

## 値

**errorcode**

エラーのタイプを示します。

## 説明

これは NFSv4 のエラー処理ルーチンです。

---

## 名前

probe::nfs.proc.lookup – NFS クライアントはサーバー上のファイルを開き、検索します。

## 概要

```
nfs.proc.lookup
```

## 値

**bitmask1**

このファイルシステムでサポートされる属性セットを表す V4 ビットマスク。

**bitmask0**

このファイルシステムでサポートされる属性セットを表す V4 ビットマスク。

**filename**

クライアントがサーバー上で開き、検索するファイルの名前。

**server\_ip**

サーバーの IP アドレス。

**prot**

転送プロトコル。



**name\_len**

ファイル名の長さ。

**version**

NFS バージョン。

---

**名前**

**probe::nfs.proc.open** – NFS クライアントはファイルの読み取り/書き込みコンテキスト情報を割り当てます。

**概要**

```
nfs.proc.open
```

**値****flag**

ファイルフラグ。

**filename**

ファイル名。

**version**

NFS バージョン (この関数はすべての NFS バージョンに使用されます)。

**prot**

転送プロトコル。

**mode**

ファイルモード。

**server\_ip**

サーバーの IP アドレス。

**説明**

読み取り/書き込みコンテキスト情報を割り当てます。

---

**名前**

**probe::nfs.proc.read** – NFS クライアントはサーバー上のファイルを同期的に読み取ります。

**概要**

**nfs.proc.read**

## 値

### **offset**

ファイルオフセット。

### **server\_ip**

サーバーの IP アドレス。

### **flags**

`rpc_init_task` 関数で `task->tk_flags` を設定するために使用されます。

### **prot**

転送プロトコル。

### **count**

この実行での読み取りバイト。

### **version**

NFS バージョン。

## 説明

2006 年 12 月にすべての `nfs.proc.read` kernel 関数は `kernel commit 8e0969` で削除されました。そのため、これらのプローブは Linux 2.6.21 以降のカーネルには存在しません。

---

## 名前

`probe::nfs.proc.read_done` – 読み取り RPC タスクに応答する NFS クライアント。

## 概要

**nfs.proc.read\_done**

## 値

### **timestamp**

リリースの更新に使用される V4 タイムスタンプ。

### **prot**

転送プロトコル。

### **count**

読み取りバイト数。

### **version**

NFS バージョン。

**status**

最終操作の結果。

**server\_ip**

サーバーの IP アドレス。

## 説明

読み取り RPC タスクへの返信が受信されるか、一部の読み取り操作のエラー (タイムアウトまたはソケットのシャットダウン) が発生する際に実行されます。

---

## 名前

probe::nfs.proc.read\_setup – 読み取り RPC タスクをセットアップする NFS クライアント。

## 概要

```
nfs.proc.read_setup
```

## 値

**offset**

ファイルオフセット。

**server\_ip**

サーバーの IP アドレス。

**prot**

転送プロトコル。

**version**

NFS バージョン。

**count**

この実行での読み取りバイト。

**size**

この実行での読み取りバイト。

## 説明

`read_setup` 関数は読み取り RPC タスクをセットアップするために使用されます。これは実際の読み取り操作を実行しません。

---

## 名前

**probe::nfs.proc.release** – NFS クライアントはファイルの読み取り/書き込みコンテキスト情報をリリースします。

## 概要

**nfs.proc.release**

## 値

### **flag**

ファイルフラグ。

### **filename**

ファイル名。

### **prot**

転送プロトコル。

### **version**

NFS バージョン (この関数はすべての NFS バージョンに使用されます)。

### **mode**

ファイルモード。

### **server\_ip**

サーバーの IP アドレス。

## 説明

読み取り/書き込みコンテキスト情報をリリースします。

---

## 名前

**probe::nfs.proc.remove** – NFS クライアントはサーバー上のファイルを削除します。

## 概要

**nfs.proc.remove**

## 値

### **prot**

転送プロトコル。

### **version**

NFS バージョン (この関数はすべての NFS バージョンに使用されます)。

**server\_ip**

サーバーの IP アドレス。

**filelen**

ファイル名の長さ。

**filename**

ファイル名。

**fh**

親ディレクトリーのファイルハンドル。

---

## 名前

`probe::nfs.proc.rename` – NFS クライアントはサーバー上のファイルの名前を変更します。

## 概要

```
nfs.proc.rename
```

## 値

**new\_fh**

新規の親ディレクトリーのファイルハンドル。

**new\_filelen**

新規ファイル名の長さ。

**old\_name**

古いファイルの名前。

**version**

NFS バージョン (この関数はすべての NFS バージョンに使用されます)。

**old\_fh**

古い親ディレクトリーのファイルハンドル。

**prot**

転送プロトコル。

**new\_name**

新しいファイルの名前。

**old\_filelen**

古いファイル名の長さ。

**server\_ip**

サーバーの IP アドレス。

---

**名前**

probe::nfs.proc.rename\_done – 名前変更 RPC タスクに対応する NFS クライアント。

**概要**

`nfs.proc.rename_done`

**値****timestamp**

リリースの更新に使用される V4 タイムスタンプ。

**status**

最終操作の結果。

**server\_ip**

サーバーの IP アドレス。

**prot**

転送プロトコル。

**version**

NFS バージョン。

**old\_fh**

古い親ディレクトリーのファイルハンドル。

**new\_fh**

新規の親ディレクトリーのファイルハンドル。

**説明**

名前変更 RPC タスクへの返信が受信されるか、一部の読み取り操作のエラー (タイムアウトまたはソケットのシャットダウン) が発生する際に実行されます。

---

**名前**

probe::nfs.proc.rename\_setup – 名前変更 RPC タスクをセットアップする NFS クライアント

**概要**

`nfs.proc.rename_setup`

値

***fh***

親ディレクトリーのファイルハンドル。

***prot***

転送プロトコル。

***version***

NFS バージョン。

***server\_ip***

サーバーの IP アドレス。

説明

`rename_setup` 関数は読み取り RPC タスクをセットアップするために使用されます。これは実際の名前変更操作を実行しません。

名前

`probe::nfs.proc.write` – NFS クライアントはサーバーにファイルを同期的に書き込みます。

概要

`nfs.proc.write`

値

***size***

この実行での読み取りバイト。

***flags***

`rpc_init_task` 関数で `task->tk_flags` を設定するために使用されます。

***prot***

転送プロトコル。

***version***

NFS バージョン。

***bitmask1***

このファイルシステムでサポートされる属性セットを表す V4 ビットマスク。

***offset***

ファイルオフセット。

**bitmask0**

このファイルシステムでサポートされる属性セットを表す V4 ビットマスク。

**server\_ip**

サーバーの IP アドレス。

## 説明

2006 年 12 月にすべての `nfs.proc.write` カーネル関数は `kernel commit 200baa` で削除されました。そのため、これらのプローブは Linux 2.6.21 以降のカーネルには存在しません。

---

## 名前

`probe::nfs.proc.write_done` – 書き込み RPC タスクに応答する NFS クライアント

## 概要

`nfs.proc.write_done`

## 値

**server\_ip**

サーバーの IP アドレス。

**status**

最終操作の結果。

**version**

NFS バージョン。

**count**

書き込みバイト数

**prot**

転送プロトコル。

**valid**

`fattr->valid` は有効なファイルを示します。

**timestamp**

リリースの更新に使用される V4 タイムスタンプ。

## 説明



書き込み RPC タスクへの返信が受信されるか、一部の書き込み操作のエラー (タイムアウトまたはソケットのシャットダウン) が発生する際に実行されます。

---

## 名前

`probe::nfs.proc.write_setup` – 書き込み RPC タスクをセットアップする NFS クライアント

## 概要

`nfs.proc.write_setup`

## 値

### *size*

この実行で書き込まれたバイト数

### *prot*

転送プロトコル。

### *version*

NFS バージョン。

### *count*

この実行で書き込まれたバイト数

### *bitmask0*

このファイルシステムでサポートされる属性セットを表す V4 ビットマスク。

### *bitmask1*

このファイルシステムでサポートされる属性セットを表す V4 ビットマスク。

### *offset*

ファイルオフセット。

### *how*

`args.stable` を設定するために使用されます。安定した値は、`NFS_UNSTABLE`、`NFS_DATA_SYNC`、`NFS_FILE_SYNC` (`nfs.proc3.write_setup` および `nfs.proc4.write_setup` に含まれます) です。

### *server\_ip*

サーバーの IP アドレス。

## 説明

`write_setup` 関数は書き込み RPC タスクをセットアップするために使用されます。これは実際の書き込み操作を実行しません。

---

## 名前

`probe::nfsd.close` – クライアントのファイルを閉じる NFS サーバー

## 概要

`nfsd.close`

## 値

### ***filename***

ファイル名。

## 説明

このプローブポイントは、4.2 以降のカーネルには存在しません。

---

## 名前

`probe::nfsd.commit` – 保留中のすべての書き込みを安定したストレージにコミットする NFS サーバー

## 概要

`nfsd.commit`

## 値

### ***fh***

ファイルハンドル (最初の部分はファイルハンドルの長さ)

### ***flag***

この実行が同期操作であるかどうかを示します。

### ***offset***

ファイルのオフセット

### ***size***

読み取りサイズ。

### ***count***

読み取りサイズ。

### ***client\_ip***

クライアントの IP アドレス

---

## 名前

`probe::nfsd.create` – クライアント用ファイル (`regular`、`dir`、`device`、`fifo`) を作成する NFS サーバー

## 概要

`nfsd.create`

## 値

### *fh*

ファイルハンドル (最初の部分はファイルハンドルの長さ)

### *iap\_valid*

属性フラグ

### *filelen*

ファイル名の長さ。

### *type*

ファイルタイプ (`regular`、`dir`、`device`、`fifo` など)

### *filename*

ファイル名。

### *iap\_mode*

ファイルアクセスモード

### *client\_ip*

クライアントの IP アドレス

## 説明

`nfsd` は、このプローブポイントの代わりに `nfsd_create_v3` を呼び出すことがあります。

---

## 名前

`probe::nfsd.createv3` – クライアント用の通常のファイルを作成、またはファイル属性を設定する NFS サーバー

## 概要

`nfsd.createv3`

## 値

### *iap\_mode*

ファイルアクセスモード

**filename**

ファイル名。

**client\_ip**

クライアントの IP アドレス

**fh**

ファイルハンドル (最初の部分はファイルハンドルの長さ)

**createmode**

作成モード。可能な値は NFS3\_CREATE\_EXCLUSIVE、NFS3\_CREATE\_UNCHECKED、または NFS3\_CREATE\_GUARDED になります。

**filelen**

ファイル名の長さ。

**iap\_valid**

属性フラグ

**verifier**

ファイル属性 (atime、mtime、mode)。CREATE\_EXCLUSIVE のファイル属性をリセットするために使用されます。

**truncp**

truncp 引数。ファイルを切り捨てるかどうかを示します。

## 説明

このプローブポイントは、`op_claim_type` が `NFS4_OPEN_CLAIM_NULL` の場合に `nfsd3_proc_create` と `nfsd4_open` によってのみ呼び出されます。

---

## 名前

`probe::nfsd.dispatch` – NFS サーバーはクライアントから操作を受け取ります。

## 概要

`nfsd.dispatch`

## 値

**xid**

送信 ID

**version**

NFS バージョン

**proto**

転送プロトコル。

**proc**

プロシージャ－番号

**client\_ip**

クライアントの IP アドレス

**prog**

プログラム番号

---

**名前**

probe::nfsd.lookup－クライアント用のファイルを開く、または検索する NFS サーバー

**概要**

nfsd.lookup

**値****filename**

ファイル名。

**client\_ip**

クライアントの IP アドレス

**fh**

親ディレクトリーのファイルハンドル (最初の部分はファイルハンドルの長さ)

**filelen**

ファイル名の長さ。

---

**名前**

probe::nfsd.open－クライアント用ファイルを開く NFS サーバー

**概要**

nfsd.open

**値****fh**

...

ファイルハンドル (最初の部分はファイルハンドルの長さ)

**type**

ファイルのタイプ (通常のファイルまたはディレクトリー)

**access**

オープンのタイプを示します (read/write/commit/readdir...)

**client\_ip**

クライアントの IP アドレス

---

## 名前

probe::nfsd.proc.commit – クライアント用コミット操作を実行する NFS サーバー

## 概要

`nfsd.proc.commit`

## 値

**count**

読み取りサイズ。

**client\_ip**

クライアントの IP アドレス

**proto**

転送プロトコル。

**size**

読み取りサイズ。

**version**

NFS バージョン

**uid**

要求者のユーザー ID

**offset**

ファイルのオフセット

**gid**

要求者のグループ ID

**fh**

ファイルハンドル (最初の部分はファイルハンドルの長さ)

---

**名前**

probe::nfsd.proc.create – クライアント用ファイルを作成する NFS サーバー

**概要**

**nf**sd.proc.create

**値****proto**

転送プロトコル。

**filename**

ファイル名。

**client\_ip**

クライアントの IP アドレス

**uid**

要求者のユーザー ID

**version**

NFS バージョン

**gid**

要求者のグループ ID

**fh**

ファイルハンドル (最初の部分はファイルハンドルの長さ)

**filelen**

ファイル名の長さ。

---

**名前**

probe::nfsd.proc.lookup – クライアント用のファイルを開く、または検索する NFS サーバー

**概要**

**nf**sd.proc.lookup

## 値

### ***fh***

親ディレクトリーのファイルハンドル (最初の部分はファイルハンドルの長さ)

### ***gid***

要求者のグループ ID

### ***filelen***

ファイル名の長さ。

### ***uid***

要求者のユーザー ID

### ***version***

NFS バージョン

### ***proto***

転送プロトコル。

### ***filename***

ファイル名。

### ***client\_ip***

クライアントの IP アドレス

---

## 名前

`probe::nfsd.proc.read` – クライアント用ファイルを読み取る NFS サーバー

## 概要

`nfsd.proc.read`

## 値

### ***size***

読み取りサイズ。

### ***vec***

構造 `kvec`。カーネルアドレスの `buf` アドレスと各バッファの長さを含みます。

### ***version***

NFS バージョン



**uid**

要求者のユーザー ID

**count**

読み取りサイズ。

**client\_ip**

クライアントの IP アドレス

**proto**

転送プロトコル。

**offset**

ファイルのオフセット

**gid**

要求者のグループ ID

**vlen**

ブロックの読み取り

**fh**

ファイルハンドル (最初の部分はファイルハンドルの長さ)

---

## 名前

probe::nfsd.proc.remove – クライアント用ファイルを削除する NFS サーバー

## 概要

**■** nfsd.proc.remove

## 値

**gid**

要求者のグループ ID

**fh**

ファイルハンドル (最初の部分はファイルハンドルの長さ)

**filelen**

ファイル名の長さ。

**uid**

要求者のユーザー ID

**version**

NFS バージョン

**proto**

転送プロトコル。

**filename**

ファイル名。

**client\_ip**

クライアントの IP アドレス

---

## 名前

probe::nfsd.proc.rename – クライアント用ファイルの名前を変更する NFS サーバー

## 概要

```
nfsd.proc.rename
```

## 値

**uid**

要求者のユーザー ID

**tfh**

新しいパスのファイルハンドラー

**tname**

新しいファイルの名前。

**filename**

古いファイルの名前。

**client\_ip**

クライアントの IP アドレス

**flen**

古いファイル名の長さ。

**gid**

要求者のグループ ID

**fh**

古いパスのファイルハンドラー

***tlen***

新規ファイル名の長さ。

---

**名前**

probe::nfsd.proc.write – クライアント用ファイルにデータを書き込む NFS サーバー

**概要**

**|** nfsd.proc.write

**値*****offset***

ファイルのオフセット

***gid***

要求者のグループ ID

***vlen***

ブロックの読み取り

***fh***

ファイルハンドル (最初の部分はファイルハンドルの長さ)

***size***

読み取りサイズ。

***vec***

構造 kvec。カーネルアドレスの buf アドレスと各バッファの長さを含みます。

***stable***

argp->stable

***version***

NFS バージョン

***uid***

要求者のユーザー ID

***count***

読み取りサイズ。

***client\_ip***

クライアントの IP アドレス

**proto**

転送プロトコル。

---

**名前**

`probe::nfsd.read` – クライアント用ファイルからデータを読み取る NFS サーバー

**概要**

`nfsd.read`

**値****offset**

ファイルのオフセット

**vlen**

ブロックの読み取り

**file**

引数ファイル。ファイルがオープンであるかどうかを示します。

**fh**

ファイルハンドル (最初の部分はファイルハンドルの長さ)

**count**

読み取りサイズ。

**client\_ip**

クライアントの IP アドレス

**size**

読み取りサイズ。

**vec**

構造 `kvec`。カーネルアドレスの `buf` アドレスと各バッファの長さを含みます。

---

**名前**

`probe::nfsd.rename` – クライアント用ファイルの名前を変更する NFS サーバー

**概要**

`nfsd.rename`

## 値

### ***tlen***

新規ファイル名の長さ。

### ***fh***

古いパスのファイルハンドラー

### ***flen***

古いファイル名の長さ。

### ***client\_ip***

クライアントの IP アドレス

### ***filename***

古いファイルの名前。

### ***tname***

新しいファイルの名前。

### ***tfh***

新しいパスのファイルハンドラー

---

## 名前

`probe::nfsd.unlink` – クライアント用ファイルまたはディレクトリーを削除する NFS サーバー

## 概要

`nfsd.unlink`

## 値

### ***filelen***

ファイル名の長さ。

### ***fh***

ファイルハンドル (最初の部分はファイルハンドルの長さ)

### ***type***

ファイルの種類 (ファイルまたはディレクトリー)

### ***client\_ip***

クライアントの IP アドレス

**filename**

ファイル名。

---

**名前**

probe::nfsd.write – クライアント用ファイルにデータを書き込む NFS サーバー

**概要**

**■** nfsd.write

**値****offset**

ファイルのオフセット

**fh**

ファイルハンドル (最初の部分はファイルハンドルの長さ)

**vlen**

ブロックの読み取り

**file**

引数ファイル。ファイルがオープンであるかどうかを示します。

**client\_ip**

クライアントの IP アドレス

**count**

読み取りサイズ。

**size**

読み取りサイズ。

**vec**

構造 **kvec**。カーネルアドレスの **buf** アドレスと各バッファの長さを含みます。

## 第33章 予測

この種類の関数は、予測して情報を記録し、**SystemTap** スクリプトの後半で情報をコミットするか、破棄する機能を提供します。

### 名前

**function::commit** – 予測バッファーに関連するすべての出力を書き出します。

### 概要

```
commit(id:long)
```

### 引数

**id**

情報を格納するバッファー

### 説明

**speculative** により予測バッファーに入力された順序で **id** のすべての出力を出力します。

---

### 名前

**function::discard** – 予測バッファーに関連するすべての出力を破棄します。

### 概要

```
discard(id:long)
```

### 引数

**id**

情報を格納するバッファー

---

### 名前

**function::speculate** – 後で出力する可能性がある文字列を格納します。

### 概要

```
speculate(id:long,output:string)
```

### 引数

**id**

情報を格納するバッファー ID

---

**output**

コミットが実行されたときに書き出す文字列

**説明**

ID のために文字列を予測バッファに追加します。

---

**名前**

**function::speculation** – 予測出力のために新しい ID を割り当てます。

**概要**

```
speculation:long()
```

**引数**

なし

**説明**

**speculation** 関数は、新しい予測バッファが必要な場合に呼び出され、予測出力のために ID を返します。複数のスレッドを同時に予測できます。この ID は、スレッドを区別するために他の予測関数によって使用されます。



## 第34章 JSON タップセット

この種類のプローブポイント、関数、およびマクロは、JSON 形式でデータを出力するために使用されます。以下のプローブポイント、関数、およびマクロが含まれます。

### 名前

function::json\_add\_array – アレイの追加

### 概要

```
json_add_array:long(name:string,description:string)
```

### 引数

#### *name*

アレイの名前

#### *description*

アレイの説明。空の文字列を使用できます。

### 説明

この関数は、アレイを追加し、必要なことをすべてセットアップします。アレイには、**json\_add\_array\_numeric\_metric** または **json\_add\_array\_string\_metric** で追加された他のメトリックスが含まれます。

---

### 名前

function::json\_add\_array\_numeric\_metric – 数値メトリックをアレイに追加します。

### 概要

```
json_add_array_numeric_metric:long(array_name:string,metric_name:string,metric_description:string,metric_units:string)
```

### 引数

#### *array\_name*

数値メトリックを追加する必要があるアレイの名前

#### *metric\_name*

数値メトリックの名前

#### *metric\_description*

メトリックの説明。空の文字列を使用できます。

#### *metric\_units*

メトリック単位。空の文字列を使用できます。

## 説明

この関数は、アレイに数値メトリックを追加し、必要なことをすべてセットアップします。

---

## 名前

**function::json\_add\_array\_string\_metric** – 文字列メトリックをアレイに追加します。

## 概要

```
json_add_array_string_metric:long(array_name:string,metric_name:string,metric_description:string)
```

## 引数

### ***array\_name***

文字列メトリックを追加する必要があるアレイの名前

### ***metric\_name***

文字列メトリックの名前

### ***metric\_description***

メトリックの説明。空の文字列を使用できます。

## 説明

この関数は、アレイに文字列メトリックを追加し、必要なことをすべてセットアップします。

---

## 名前

**function::json\_add\_numeric\_metric** – 数値メトリックを追加します。

## 概要

```
json_add_numeric_metric:long(name:string,description:string,units:string)
```

## 引数

### ***name***

数値メトリックの名前

### ***description***

メトリックの説明。空の文字列を使用できます。

### ***units***

メトリック単位。空の文字列を使用できます。

## 説明

この関数は、数値メトリックを追加し、必要なことをすべてセットアップします。

---

## 名前

**function::json\_add\_string\_metric** – 文字列メトリックを追加します。

## 概要

```
json_add_string_metric:long(name:string,description:string)
```

## 引数

### **name**

文字列メトリックの名前

### **description**

メトリックの説明。空の文字列を使用できます。

## 説明

この関数は、文字列メトリックを追加し、必要なことをすべてセットアップします。

---

## 名前

**function::json\_set\_prefix** – メトリック接頭辞を設定します。

## 概要

```
json_set_prefix:long(prefix:string)
```

## 引数

### **prefix**

使用する接頭辞名。

## 説明

この関数は、メトリック階層のベースの名前である「**prefix**」を設定します。この関数の呼び出しはオプションであり、デフォルトでは、システムタップモジュールの名前が使用されます。

---

## 名前

**macro::json\_output\_array\_numeric\_value** – アレイのメトリックの数値を出力します。

## 概要

```
@json_output_array_numeric_value(array_name,array_index,metric_name,value)
```

## 引数

### ***array\_name***

アレイの名前

### ***array\_index***

数値を格納する場所を示すアレイインデックス (文字列)。

### ***metric\_name***

数値メトリックの名前

### ***value***

出力する数値。

## 説明

`json_output_array_numeric_value` マクロは、アレイ内にあるメトリックの数値を出力するためにユーザーのスクリプトの `'json_data'` プロープから呼び出されるよう設計されています。このメトリックは、`json_add_array_numeric_metric` で追加されているはずです。

---

## 名前

`macro::json_output_array_string_value` – アレイのメトリックの文字列値を出力します。

## 概要

```
@json_output_array_string_value(array_name,array_index,metric_name,value)
```

## 引数

### ***array\_name***

アレイの名前

### ***array\_index***

文字列値を格納する場所を示すアレイインデックス (文字列)。

### ***metric\_name***

文字列メトリックの名前

### ***value***

出力する文字列値。

## 説明

`json_output_array_string_value` マクロは、アレイ内にあるメトリックの文字列値を出力するためにユーザーのスクリプトの 'json\_data' プロープから呼び出されるよう設計されています。このメトリックは、`json_add_array_string_metric` で追加されているはずです。

---

## 名前

`macro::json_output_data_end` – json 出力を終了します。

## 概要

```
@json_output_data_end()
```

## 引数

なし

## 説明

`json_output_data_end` マクロは、ユーザーのスクリプトの 'json\_data' プロープから呼び出されるよう設計されています。JSON 出力の最後をマークします。

---

## 名前

`macro::json_output_data_start` – json 出力を開始します。

## 概要

```
@json_output_data_start()
```

## 引数

なし

## 説明

`json_output_data_start` マクロは、ユーザーのスクリプトの 'json\_data' プロープから呼び出されるよう設計されています。JSON 出力の開始をマークします。

---

## 名前

`macro::json_output_numeric_value` – 数値を出力します。

## 概要

```
@json_output_numeric_value(name, value)
```

## 引数

***name***

数値メトリックの名前

**value**

出力する数値。

## 説明

`json_output_numeric_value` マクロは、メトリックの数値を出力するためにユーザーのスクリプトの `'json_data'` プローブから呼び出されるよう設計されています。このメトリックは、`json_add_numeric_metric` で追加されているはずです。

---

## 名前

`macro::json_output_string_value` – 文字列値を出力します。

## 概要

```
@json_output_string_value(name,value)
```

## 引数

**name**

文字列メトリックの名前

**value**

出力する文字列値。

## 説明

`json_output_string_value` マクロは、メトリックの文字列値を出力するためにユーザーのスクリプトの `'json_data'` プローブから呼び出されるよう設計されています。このメトリックは、`json_add_string_metric` で追加されているはずです。

---

## 名前

`probe::json_data` – JSON データをリーダーが必要とするときに必ず実行されます。

## 概要

```
json_data
```

## 値

なし

## コンテキスト

このプロブは、JSON データが読み取られるときに実行されます。このプロブはデータを収集し、以下のマクロを呼び出して JSON 形式でデータを出力する必要があります。最初に、`@json_output_data_start` を呼び出す必要があります。この呼び出しの後に、次の1つまたは複数の呼び出しが続きます (各データアイテムに対して1つの呼び出し):

@json\_output\_string\_value、@json\_output\_numeric\_value、  
@json\_output\_array\_string\_value、および @json\_output\_array\_numeric\_value。最後に、@json\_output\_data\_end を呼び出す必要があります。

## 第35章 出力ファイル切り替えタップセット

出力ファイルの切り替えを許可するユーティリティー関数

### 名前

**function::switch\_file** – 次の出力ファイルへの切り替え

### 概要

```
switch_file()
```

### 引数

なし

### 説明

この関数は、**stapio** プロセスにシグナルを送信し、出力がファイルに送信されたときに次の出力ファイルにローテーションするよう指示します。



## 付録A 改訂履歴

<b>改訂 1-4.6</b> 翻訳ファイルを XML ソースバージョン 1-4 と同期	<b>Tue Jul 4 2017</b>	<b>Terry Chuang</b>
<b>改訂 1-4.5</b> Translation Completed	<b>Thu Mar 9 2017</b>	<b>Terry Chuang</b>
<b>改訂 1-4.4</b> 翻訳ファイルを XML ソースバージョン 1-4 と同期	<b>Thu Mar 9 2017</b>	<b>Terry Chuang</b>
<b>改訂 1-4.3</b> 翻訳ファイルを XML ソースバージョン 1-4 と同期	<b>Mon Mar 6 2017</b>	<b>Terry Chuang</b>
<b>改訂 1-4.2</b> 翻訳ファイルを XML ソースバージョン 1-4 と同期	<b>Mon Mar 6 2017</b>	<b>Terry Chuang</b>
<b>改訂 1-4.1</b> 翻訳ファイルを XML ソースバージョン 1-4 と同期	<b>Wed Mar 1 2017</b>	<b>Terry Chuang</b>
<b>改訂 1-4</b> Red Hat Enterprise Linux 7.3 用のリリース	<b>Wed Oct 19 2016</b>	<b>Robert Krátký</b>
<b>改訂 1-2</b> Red Hat Enterprise Linux 7.3 用の非同期リリース	<b>Thu Mar 10 2016</b>	<b>Robert Kratky</b>
<b>改訂 1-2</b> Red Hat Enterprise Linux 7.3 用のリリース	<b>Thu Nov 11 2015</b>	<b>Robert Kratky</b>