ChorusOS man pages section 2MON: Monitoring Services

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900
U.S.A.

Part No: 806-3327
December 10, 1999

Contents

PREFACE

Overview

A man page is provided for both the naive user, and sophisticated user who is familiar with the ChorusOS™ operating system and is in need of on-line information. A man page is intended to answer concisely the question “What does it do?” The man pages in general comprise a reference manual. They are not intended to be a tutorial.

The following is a list of sections in the ChorusOS man pages and the information it references:

- *Section 1CC: User Utilities; Host and Target Utilities*
- *Section 1M: System Management Utilities*
- *Section 2DL: System Calls; Data Link Services*
- *Section 2K: System Calls; Kernel Services*
- *Section 2MON: System Calls; Monitoring Services*
- *Section 2POSIX: System Calls; POSIX System Calls*
- *Section 2RESTART: System Calls; Hot Restart and Persistent Memory*
- *Section 2SEG: System Calls; Virtual Memory Segment Services*
- *Section 3FTPD: Libraries; FTP Daemon*
- *Section 3M: Libraries; Mathematical Libraries*
- *Section 3POSIX: Libraries; POSIX Library Functions*
- *Section 3RPC: Libraries; RPC Services*
- *Section 3STDC: Libraries; Standard C Library Functions*
- *Section 3TELD: Libraries; Telnet Services*
- *Section 4CC: Files*

- *Section 5FEA: ChorusOS Features and APIs*
- *Section 7P: Protocols*
- *Section 7S: Services*
- *Section 9DDI: Device Driver Interfaces*
- *Section 9DKI: Driver to Kernel Interface*
- *Section 9DRV: Driver Implementations*

ChorusOS man pages are grouped in Reference Manuals, with one reference manual per section.

Below is a generic format for man pages. The man pages of each manual section generally follow this order, but include only needed headings. For example, if there are no bugs to report, there is no BUGS section. See the `intro` pages for more information and detail about each section, and `man`(1) for more information about man pages in general.

NAME — This section gives the names of the commands or functions documented, followed by a brief description of what they do.

SYNOPSIS — This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full pathname is shown. Options and arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument order is required.

The following special characters are used in this section:

[ ] The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.

. . . Ellipses. Several values may be provided for the previous argument, or the previous argument can be specified multiple times, for example, ‘ "`filename` . . ." .

| Separator. Only one of the arguments separated by this character can be specified at time.

{ } Braces. The options and / or arguments enclosed within braces are

	interdependent, such that everything enclosed must be treated as a unit.
FEATURES	This section provides the list of features which offer an interface. An API may be associated with one or more system features. The interface will be available if one of the associated features has been configured.
DESCRIPTION	This section defines the functionality and behavior of the service. Thus it describes concisely what the command does. It does not discuss OPTIONS or cite EXAMPLES.. Interactive commands, subcommands, requests, macros, functions and such, are described under USAGE.
OPTIONS	This lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.
OPERANDS	This section lists the command operands and describes how they affect the actions of the command.
OUTPUT	This section describes the output - standard output, standard error, or output files - generated by the command.
RETURN VALUES	If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or –1, these values are listed in tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared void do not return values, so they are not discussed in RETURN VALUES.
ERRORS	On failure, most functions place an error code in the global variable `errno` indicating why they failed. This section lists alphabetically all error codes a function can generate and describes the conditions that cause each error. When more than one condition can cause the same error, each condition is described in a separate paragraph under the error code.

USAGE	This section is provided as a guidance on use. This section lists special rules, features and commands that require in-depth explanations. The subsections listed below are used to explain built-in functionality: Commands Modifiers Variables Expressions Input Grammar
EXAMPLES	This section provides examples of usage or of how to use a command or function. Wherever possible a complete example including command line entry and machine response is shown. Whenever an example is given, the prompt is shown as `example%` or if the user must be superuser, `example#`. Examples are followed by explanations, variable substitution rules, or returned values. Most examples illustrate concepts from the SYNOPSIS, DESCRIPTION, OPTIONS and USAGE sections.
ENVIRONMENT VARIABLES	This section lists any environment variables that the command or function affects, followed by a brief description of the effect.
EXIT STATUS	This section lists the values the command returns to the calling program or shell and the conditions that cause these values to be returned. Usually, zero is returned for successful completion and values other than zero for various error conditions.
FILES	This section lists all filenames referred to by the man page, files of interest, and files created or required by commands. Each is followed by a descriptive summary or explanation.
SEE ALSO	This section lists references to other man pages, in-house documentation and outside publications.
DIAGNOSTICS	This section lists diagnostic messages with a brief explanation of the condition causing the error.
WARNINGS	This section lists warnings about special conditions which could seriously affect your working conditions. This is not a list of diagnostics.
NOTES	This section lists additional information that does not belong anywhere else on the page. It takes the form of an aside to the user, covering points of special interest. Critical information is never covered here.

BUGS

This section describes known bugs and wherever possible, suggests workarounds.

CHAPTER

Monitoring Services

10

NAME

KcModule, UcModule – Array of all micro-kernel supervisor modules and system calls; Array of all micro-kernel user modules and system calls

SYNOPSIS

```
#include <mon/chModules.h>
extern KernelModule KcModule [K_CMODULE_MAX];
extern KernelModule UcModule [K_CMODULE_MAX];
```

FEATURES

MON

DESCRIPTION

The `KcModule` array contains a description of the supervisor modules and system calls available on the system. Each entry in the array is a *KernelModule* structure whose members are the following:

```
char*         kmName ;
int           kmScCnt ;
KernelCall*   kmSc ;
```

The *kmName* member specifies the name of the module. The *kmScCnt* member specifies the number of system calls defined by the module. The *kmSc* member is a pointer to an array describing the module's (supervisor) system calls. This array has *kmScCnt* entries; the n-th entry contains the description of the n-th system call of the module. The entries are of the type *KernelCall* . The *KernelCall* structure has the following members:

```
char*   kcName ;
int     kcModule ;
int     kcNo ;
int     kcArgsCnt ;
```

The *kcName* member specifies the name of the system call. The *kcModule* member specifies the module to which the system call belongs. The *kcNo* member specifies the system call's number. The *kcArgsCnt* member specifies the number of arguments used by the system call.

The `<mon/chKcalls.h>` header file defines a macro for each supervisor system call, enabling the system call to be mapped to the corresponding entry in the *KcModule* array. For example:

```
#define svTrapConnect_sc (&KcModule[0].kmSc[24])
```

The `UcModule` array contains a description of the user modules and system calls available on the system. Its layout is the same as the *KcModule* array.

The `<mon/chUcalls.h>` header file defines a macro for each user system call, enabling the system call to be mapped to the corresponding entry in the *UcModule* array. For example:

```
#define actorCreate_uc (&UcModule[0].kmSc[1])
```

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

NAME

svActorMonConst – retrieve the value of an actor's monitoring constants

SYNOPSIS

```
#include <mon/chMon.h>
int svActorMonConst(MonActorConst *constants, int size);
```

FEATURES

MON

DESCRIPTION

The *svActorMonConst* kernel call fills the *MonActorConst* structure pointed to by the *constants* parameter. The `size` parameter must be set to *sizeof(MonActorConst)*. This feature is provided for compatibility between different releases of the kernel. The members of the *MonActorConst* structure are the following:

```
int   actName_offset ;
int   actName_sizeof ;
int   actCap_offset ;
```

These specify monitoring constants related to actors, and the offsets which detail the layout of the *MonActorState* structure. The *MonActorState* structure gives the internal representation of an actor within the kernel data space, see *svActorProbeConnect*(2MON).

The *actName_offset* member is the offset of the string representing the name of the actor. The *actName_sizeof* member is the space reserved for the name. The *actCap_offset* member is the offset of the capability of the actor.

The semantics of the members of the *MonActorConst* structure is summarized by the following lines of pseudo-code:

```
MonActorState*  actorState ;
char            (actorState + actName_offset) [actName_sizeof] ;
KnCap*          (actState + actCap_offset) ;
```

The *K_ENOTIMP* value indicates that the corresponding offset or constant does not apply to this particular release of the micro-kernel.

RETURN VALUE

The return value is set to the size in bytes of the `MonActorConst` object.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`svSiteMonConst`(2MON), `svThreadMonConst`(2MON), `svPortMonConst`(2MON), `svActorProbeConnect`(2MON), `actorName`(2K)

NAME

svSiteActorList, svActorThreadList, svActorPortList – List the actors belonging to the local site; List the threads belonging to the local actor; List the ports belonging to the local actor

SYNOPSIS

```
#include <mon/chMonProbe.h>
int svSiteActorList(void);

int svActorThreadList(KnCap * actorcap);

int svActorPortList(KnCap * actorcap);
```

FEATURES

MON

DESCRIPTION

The *svSiteActorList* notifies a monitor of the list of actors belonging to the local site, via the monitoring probe (if any) connected to the local site. For each actor belonging to the local site, the system call invokes the *nextActor* notification routine (see *svSiteProbeConnect* (2MON)). Finally, the *nextActor* notification routine is invoked with a null parameter and *svSiteActorList* returns.

The *svActorThreadList* kernel call notifies a monitor of the list of threads belonging to an actor, via the monitoring probe (if any) connected to the actor. For each thread belonging to the actor, the system call invokes the *nextThread* notification routine (see *svActorProbeConnect* (2MON)). Finally, the *nextThread* notification routine is invoked with a null parameter and *svActorThreadList* returns.

The *svActorPort* kernel call notifies a monitor of the list of ports belonging to an actor, via the monitoring probe (if any) connected to this actor. For each port belonging to the actor, the system call invokes the *nextPort* notification routine (see *svActorProbeConnect* (2MON)). Finally, the *nextPort* notification routine is invoked with a null parameter and *svActorPortList* returns.

RETURN VALUE

Upon successful completion, both kernel calls return 0. Otherwise, a negative error code is returned.

ERRORS

[K_EINVAL]	*actorcap* is an inconsistent actor capability.
[K_EUNKNOWN]	*actorcap* does not specify a reachable actor.

RESTRICTIONS

For *svActorThreadList* and *svActorPortList* , the target actor and the current actor must be located on the same site.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`svSiteProbeConnect`(2MON) , `svActorProbeConnect`(2MON) , `svPortProbeConnect`(2MON) , `svThreadProbeConnect`(2MON)

NAME

svActorProbeConnect, svActorProbeDisconnect – Connect a monitoring probe to an actor; Disconnect a monitoring probe from an actor

SYNOPSIS

```
#include <mon/chMon.h>
int svActorProbeConnect(KnCap * cap, MonActorProbe * probe);

int svActorProbeDisconnect(KnCap * cap, MonActorProbe * probe);
```

FEATURES

MON

DESCRIPTION

The *svActorProbeConnect* and *svActorProbeDisconnect* kernel calls connect and disconnect a monitoring probe to or from an actor, respectively. The *cap* parameter is a pointer to the capability of the target actor. The *probe* parameter describes the monitoring probe. It is a pointer to a *MonActorProbe* structure whose member is the following:

```
MonActorVtbl*   vtbl ;
```

The *vtbl* member specifies the notification routines associated with the probe. It is a pointer to a *MonActorVtbl* structure whose members are the following:

```
int                 vtbl_sizeof ;
Actor_connection    connection ;
Actor_disconnection disconnection ;
Actor_deletion      deletion ;
Actor_nextThread    nextThread ;
Actor_nextPort      nextPort ;
```

The *vtbl_sizeof* member must be set to *sizeof(MonActorVtbl)* . It is provided for compatibility between different releases of the kernel.

The other members of the *MonActorVtbl* structure are pointers to functions. Each member must either be null, or point to the implementation of a specific notification routine associated with the probe. If non-null, the notification routine will be invoked on each occurrence of the associated event (this is described later on in this manual page). If null, no invocation is performed.

The data for the *MonActorProbe* structure, for the *MonActorVtbl* structure, and the code for the monitoring routines must belong to supervisor space and must be locked in physical memory. The notification routines are executed in supervisor execution mode. Except where explicitly stated, they must not to re-enter the kernel except via a few permitted system calls (see *svIntrConnect(2K)).*

DESCRIPTION OF THE NOTIFICATION ROUTINES

The notification routines are the following:

```
void connection (probe, state)
   MonActorProbe*   probe ;
   MonActorState*   state ;
```

The *connection* routine is invoked after the probe has been connected to an actor. The *probe* parameter is a pointer to the connected probe (passed as an argument to the *svActorProbeConnect* kernel call). The *state* parameter is a pointer to a *MonActorState* structure, which represents the actor internally within the kernel address space (see *svActorMonConst* (2MON)). This pointer is valid for the lifetime of the actor.

```
void disconnection (probe)
   MonActorProbe*   probe ;
```

The *disconnection* routine is invoked after a probe has been disconnected from an actor. The *probe* parameter is a pointer to the probe involved.

```
void deletion (probe)
   MonActorProbe*   probe ;
```

The *deletion* routine is invoked after the actor has been deleted. The *probe* parameter is a pointer to the actor's probe.

```
int nextThread (probe, state)
   MonActorProbe*   probe ;
   MonThreadState*  state ;
```

The *nextThread* routine is invoked as a consequence of the *threadStat* (2K) kernel call. There is one invocation of *nextThread* for each thread of the actor. The *probe* parameter points to the probe of the actor whose threads are being listed. The *state* parameter is a pointer to a *MonThreadStat* structure; this is the internal representation of the thread's state within the kernel address space. This pointer is valid for the lifetime of the thread and is guarantee d to remain valid while the notification routine is being executed. Finally, *nextThread* is invoked a last time with a null *state* parameter to indicate that all threads have been processed.

```
int nextPort (probe, state)
   MonActorProbe*   probe ;
   MonPortState*    state ;
```

The *nextPort* routine is invoked as a consequence of the *portList* (2K) kernel call. There is one invocation of *nextPort* for each port of the actor. The *probe* parameter points to the probe of the actor whose ports are being listed. The *state* parameter is a pointer to a *MonPortStat* structure; this is the internal representation of the port's state within the kernel address space. This pointer is valid for the lifetime of the port and is guaranteed to remain valid while the notification routine is being executed. Finally, *nextPort* is invoked a last time with a null *state* parameter to indicate that all ports have been processed.

RETURN VALUE

Upon successful completion, both kernel calls return 0. Otherwise, a negative error code is returned.

ERRORS

[K_EBUSY]	An attempt was made to connect a probe while another probe was already connected to the actor.
[K_EINVAL]	An attempt was made to disconnect a probe which was not connected to the actor.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`svActorProbeConnect`(2MON) , `svThreadProbeConnect`(2MON) , `svPortProbeConnect`(2MON) , `svActorMonConst`(2MON) , `svIntrConnect`(2K)

NAME

svActorProbeConnect, svActorProbeDisconnect – Connect a monitoring probe to an actor; Disconnect a monitoring probe from an actor

SYNOPSIS

```
#include <mon/chMon.h>
int svActorProbeConnect(KnCap * cap, MonActorProbe * probe);

int svActorProbeDisconnect(KnCap * cap, MonActorProbe * probe);
```

FEATURES

MON

DESCRIPTION

The *svActorProbeConnect* and *svActorProbeDisconnect* kernel calls connect and disconnect a monitoring probe to or from an actor, respectively. The *cap* parameter is a pointer to the capability of the target actor. The *probe* parameter describes the monitoring probe. It is a pointer to a *MonActorProbe* structure whose member is the following:

```
MonActorVtbl*   vtbl ;
```

The *vtbl* member specifies the notification routines associated with the probe. It is a pointer to a *MonActorVtbl* structure whose members are the following:

```
int                 vtbl_sizeof ;
Actor_connection    connection ;
Actor_disconnection disconnection ;
Actor_deletion      deletion ;
Actor_nextThread    nextThread ;
Actor_nextPort      nextPort ;
```

The *vtbl_sizeof* member must be set to *sizeof(MonActorVtbl)* . It is provided for compatibility between different releases of the kernel.

The other members of the *MonActorVtbl* structure are pointers to functions. Each member must either be null, or point to the implementation of a specific notification routine associated with the probe. If non-null, the notification routine will be invoked on each occurrence of the associated event (this is described later on in this manual page). If null, no invocation is performed.

The data for the *MonActorProbe* structure, for the *MonActorVtbl* structure, and the code for the monitoring routines must belong to supervisor space and must be locked in physical memory. The notification routines are executed in supervisor execution mode. Except where explicitly stated, they must not to re-enter the kernel except via a few permitted system calls (see *svIntrConnect(2K)).*

DESCRIPTION OF THE NOTIFICATION ROUTINES

The notification routines are the following:

```
void connection (probe, state)
   MonActorProbe*   probe ;
   MonActorState*   state ;
```

The *connection* routine is invoked after the probe has been connected to an actor. The *probe* parameter is a pointer to the connected probe (passed as an argument to the *svActorProbeConnect* kernel call). The *state* parameter is a pointer to a *MonActorState* structure, which represents the actor internally within the kernel address space (see *svActorMonConst* (2MON)). This pointer is valid for the lifetime of the actor.

```
void disconnection (probe)
   MonActorProbe*   probe ;
```

The *disconnection* routine is invoked after a probe has been disconnected from an actor. The *probe* parameter is a pointer to the probe involved.

```
void deletion (probe)
   MonActorProbe*   probe ;
```

The *deletion* routine is invoked after the actor has been deleted. The *probe* parameter is a pointer to the actor's probe.

```
int nextThread (probe, state)
   MonActorProbe*   probe ;
   MonThreadState*  state ;
```

The *nextThread* routine is invoked as a consequence of the *threadStat* (2K) kernel call. There is one invocation of *nextThread* for each thread of the actor. The *probe* parameter points to the probe of the actor whose threads are being listed. The *state* parameter is a pointer to a *MonThreadStat* structure; this is the internal representation of the thread's state within the kernel address space. This pointer is valid for the lifetime of the thread and is guarantee d to remain valid while the notification routine is being executed. Finally, *nextThread* is invoked a last time with a null *state* parameter to indicate that all threads have been processed.

```
int nextPort (probe, state)
   MonActorProbe*   probe ;
   MonPortState*    state ;
```

The *nextPort* routine is invoked as a consequence of the *portList* (2K) kernel call. There is one invocation of *nextPort* for each port of the actor. The *probe* parameter points to the probe of the actor whose ports are being listed. The *state* parameter is a pointer to a *MonPortStat* structure; this is the internal representation of the port's state within the kernel address space. This pointer is valid for the lifetime of the port and is guaranteed to remain valid while the notification routine is being executed. Finally, *nextPort* is invoked a last time with a null *state* parameter to indicate that all ports have been processed.

RETURN VALUE

Upon successful completion, both kernel calls return 0. Otherwise, a negative error code is returned.

ERRORS

[K_EBUSY]	An attempt was made to connect a probe while another probe was already connected to the actor.
[K_EINVAL]	An attempt was made to disconnect a probe which was not connected to the actor.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`svActorProbeConnect`(2MON) , `svThreadProbeConnect`(2MON) , `svPortProbeConnect`(2MON) , `svActorMonConst`(2MON) , `svIntrConnect`(2K)

NAME

svSiteActorList, svActorThreadList, svActorPortList – List the actors belonging to the local site; List the threads belonging to the local actor; List the ports belonging to the local actor

SYNOPSIS

```
#include <mon/chMonProbe.h>
int svSiteActorList(void);

int svActorThreadList(KnCap * actorcap);

int svActorPortList(KnCap * actorcap);
```

FEATURES

MON

DESCRIPTION

The *svSiteActorList* notifies a monitor of the list of actors belonging to the local site, via the monitoring probe (if any) connected to the local site. For each actor belonging to the local site, the system call invokes the *nextActor* notification routine (see *svSiteProbeConnect* (2MON)). Finally, the *nextActor* notification routine is invoked with a null parameter and *svSiteActorList* returns.

The *svActorThreadList* kernel call notifies a monitor of the list of threads belonging to an actor, via the monitoring probe (if any) connected to the actor. For each thread belonging to the actor, the system call invokes the *nextThread* notification routine (see *svActorProbeConnect* (2MON)). Finally, the *nextThread* notification routine is invoked with a null parameter and *svActorThreadList* returns.

The *svActorPort* kernel call notifies a monitor of the list of ports belonging to an actor, via the monitoring probe (if any) connected to this actor. For each port belonging to the actor, the system call invokes the *nextPort* notification routine (see *svActorProbeConnect* (2MON)). Finally, the *nextPort* notification routine is invoked with a null parameter and *svActorPortList* returns.

RETURN VALUE

Upon successful completion, both kernel calls return 0. Otherwise, a negative error code is returned.

ERRORS

[K_EINVAL]	*actorcap* is an inconsistent actor capability.
[K_EUNKNOWN]	*actorcap* does not specify a reachable actor.

RESTRICTIONS

For *svActorThreadList* and *svActorPortList* , the target actor and the current actor must be located on the same site.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`svSiteProbeConnect`(2MON) , `svActorProbeConnect`(2MON) , `svPortProbeConnect`(2MON) , `svThreadProbeConnect`(2MON)

NAME

svPortMonConst – retrieve the value of port monitoring constants

SYNOPSIS

#include <mon/chMon.h>
int **svPortMonConst**(MonPortConst **constants*, int *size*);

FEATURES

MON

DESCRIPTION

The *svPortMonConst* kernel call fills the *MonPortConst* structure pointed to by the *constants* parameter. The `size` parameter must be set to *sizeof(MonPortConst).* It is provided for compatibility between different releases of the kernel. The members of the *MonPortConst* structure are the following:

```
int   prtName_offset ;
int   prtName_sizeof ;
int   prtLid_offset ;
```

They specify monitoring constants related to ports, and important offsets which detail the layout of the *MonPortState* structure (the *MonPortState* structure gives the internal representation of a port within the kernel data space, see *svPortProbeConnect*(2MON)).

The *prtName_offset* member is the offset of the string representing the name of the port. The *prtName_sizeof* member is the space reserved for the name. The *prtLid_offset* member is the offset of the local identifier of the port.

The semantics of the members of the *MonPortConst* structure is summarized by the following lines of pseudo-code:

```
MonPortState*   portState ;
char            (portState + prtName_offset) [prtName_sizeof] ;
int*            (portState + prtLid_offset) ;
```

The special *K_ENOTIMP* value indicates that the corresponding offset or constant does not apply to this particular release of the micro-kernel.

RETURN VALUE

The return value is set to the size in bytes of the `MonPortConst` object.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`svSiteMonConst`(2MON), `svActorMonConst`(2MON), `svThreadMonConst`(2MON), `svPortProbeConnect`(2MON)

NAME

svPortProbeConnect, svPortProbeDisconnect – Connect a monitoring probe to a port; Disconnect a monitoring probe from a port

SYNOPSIS

```
#include <mon/chMon.h>
int svPortProbeConnect(KnCap * actorcap, KnPortLid portli, MonPortProbe * probe);

int svPortProbeDisconnect(KnCap * actorcap, KnPortLid portli, MonPortProbe * probe);
```

FEATURES

MON

DESCRIPTION

The *svPortProbeConnect* and *svPortProbeDisconnect* functions respectively, connect and disconnect a monitoring probe to and from a port. The *portli* parameter is the local identifier of the target port. The *actorcap* parameter is a pointer to the capability of the target port's actor. The *probe* parameter describes the monitoring probe. It is a pointer to a *MonPortProbe* structure whose member is the following:

```
MonPortVtbl*   vtbl ;
```

The *vtbl* member specifies the notification routines associated with the probe. It is a pointer to a *MonPortVtbl* structure whose members are the following:

```
int                 vtbl_sizeof ;
Port_connection     connection ;
Port_disconnection  disconnection ;
Port_deletion       deletion ;
```

The *vtbl_sizeof* member must be set to *sizeof(MonPortVtbl)* . It is provided for compatibility between different releases of the kernel.

The other members of the *MonPortVtbl* structure are pointers to functions. Each member must either be null, or point to the implementation of a specific notification routine associated with the probe. If non null, the notification routine will be invoked on each occurrence of the associated event (this is described later on in this manual page). If null, no invocation is done.

The data for the *MonPortProbe* structure, for the *MonPortVtbl* structure, and the code for the monitoring routines must belong to supervisor space and must be locked in physical memory. The notification routines are executed in supervisor execution mode. Except where explicitly stated, they must not to re-enter the kernel except via a few permitted system calls (for example, INTERRUPTIONS).

DESCRIPTION OF THE NOTIFICATION ROUTINES

The notification routines are the following:

```
void connection (probe, portstate)
   MonPortProbe*   probe ;
   MonPortState*   portstate ;
```

The *connection* routine is invoked after the probe has been connected to a port. The *probe* parameter is a pointer to the connected probe (passed as an argument

to the *svPortProbeConnect* kernel call). The *state* parameter is a pointer to a *MonPortState* structure, which represents the actor internally within the kernel address space (see *svMonPortConst* (2MON)). This pointer is valid for the lifetime of the port.

```
void   disconnection (probe)
   MonPortProbe*   probe ;
```

The *disconnection* routine is invoked after a probe has been disconnected from a port. The *probe* parameter is a pointer to the probe concerned.

```
void   deletion (probe)
   MonPortProbe*   probe ;
```

The *deletion* routine is invoked after the actor has been deleted. The *probe* parameter is a pointer to the port's probe.

RETURN VALUE

Upon successful completion, both kernel calls return 0. Otherwise, a negative error code is returned.

ERRORS

[K_EBUSY]	An attempt was made to connect a probe while another probe was already connected to the port.
[K_EINVAL]	An attempt was made to disconnect a probe which was not connected to the port.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`svSiteProbeConnect`(2MON) , `svActorProbeConnect`(2MON) , `svThreadProbeConnect`(2MON) , `svMonPortConst`(2MON)

NAME

svPortProbeConnect, svPortProbeDisconnect – Connect a monitoring probe to a port; Disconnect a monitoring probe from a port

SYNOPSIS

#include <mon/chMon.h>
int **svPortProbeConnect**(KnCap * *actorcap*, KnPortLid *portli*, MonPortProbe * *probe*);

int **svPortProbeDisconnect**(KnCap * *actorcap*, KnPortLid *portli*, MonPortProbe * *probe*);

FEATURES

MON

DESCRIPTION

The *svPortProbeConnect* and *svPortProbeDisconnect* functions respectively, connect and disconnect a monitoring probe to and from a port. The *portli* parameter is the local identifier of the target port. The *actorcap* parameter is a pointer to the capability of the target port's actor. The *probe* parameter describes the monitoring probe. It is a pointer to a *MonPortProbe* structure whose member is the following:

```
MonPortVtbl*   vtbl ;
```

The *vtbl* member specifies the notification routines associated with the probe. It is a pointer to a *MonPortVtbl* structure whose members are the following:

```
int                 vtbl_sizeof ;
Port_connection     connection ;
Port_disconnection  disconnection ;
Port_deletion       deletion ;
```

The *vtbl_sizeof* member must be set to *sizeof(MonPortVtbl)* . It is provided for compatibility between different releases of the kernel.

The other members of the *MonPortVtbl* structure are pointers to functions. Each member must either be null, or point to the implementation of a specific notification routine associated with the probe. If non null, the notification routine will be invoked on each occurrence of the associated event (this is described later on in this manual page). If null, no invocation is done.

The data for the *MonPortProbe* structure, for the *MonPortVtbl* structure, and the code for the monitoring routines must belong to supervisor space and must be locked in physical memory. The notification routines are executed in supervisor execution mode. Except where explicitly stated, they must not to re-enter the kernel except via a few permitted system calls (for example, INTERRUPTIONS).

DESCRIPTION OF THE NOTIFICATION ROUTINES

The notification routines are the following:

```
void connection (probe, portstate)
   MonPortProbe*   probe ;
   MonPortState*   portstate ;
```

The *connection* routine is invoked after the probe has been connected to a port. The *probe* parameter is a pointer to the connected probe (passed as an argument

to the *svPortProbeConnect* kernel call). The *state* parameter is a pointer to a *MonPortState* structure, which represents the actor internally within the kernel address space (see *svMonPortConst* (2MON)). This pointer is valid for the lifetime of the port.

```
void   disconnection (probe)
   MonPortProbe*   probe ;
```

The *disconnection* routine is invoked after a probe has been disconnected from a port. The *probe* parameter is a pointer to the probe concerned.

```
void   deletion (probe)
   MonPortProbe*   probe ;
```

The *deletion* routine is invoked after the actor has been deleted. The *probe* parameter is a pointer to the port's probe.

RETURN VALUE

Upon successful completion, both kernel calls return 0. Otherwise, a negative error code is returned.

ERRORS

[K_EBUSY]	An attempt was made to connect a probe while another probe was already connected to the port.
[K_EINVAL]	An attempt was made to disconnect a probe which was not connected to the port.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`svSiteProbeConnect`(2MON) , `svActorProbeConnect`(2MON) , `svThreadProbeConnect`(2MON) , `svMonPortConst`(2MON)

NAME

svSiteActorList, svActorThreadList, svActorPortList – List the actors belonging to the local site; List the threads belonging to the local actor; List the ports belonging to the local actor

SYNOPSIS

```
#include <mon/chMonProbe.h>
int svSiteActorList(void);

int svActorThreadList(KnCap * actorcap);

int svActorPortList(KnCap * actorcap);
```

FEATURES

MON

DESCRIPTION

The *svSiteActorList* notifies a monitor of the list of actors belonging to the local site, via the monitoring probe (if any) connected to the local site. For each actor belonging to the local site, the system call invokes the *nextActor* notification routine (see *svSiteProbeConnect* (2MON)). Finally, the *nextActor* notification routine is invoked with a null parameter and *svSiteActorList* returns.

The *svActorThreadList* kernel call notifies a monitor of the list of threads belonging to an actor, via the monitoring probe (if any) connected to the actor. For each thread belonging to the actor, the system call invokes the *nextThread* notification routine (see *svActorProbeConnect* (2MON)). Finally, the *nextThread* notification routine is invoked with a null parameter and *svActorThreadList* returns.

The *svActorPort* kernel call notifies a monitor of the list of ports belonging to an actor, via the monitoring probe (if any) connected to this actor. For each port belonging to the actor, the system call invokes the *nextPort* notification routine (see *svActorProbeConnect* (2MON)). Finally, the *nextPort* notification routine is invoked with a null parameter and *svActorPortList* returns.

RETURN VALUE

Upon successful completion, both kernel calls return 0. Otherwise, a negative error code is returned.

ERRORS

[K_EINVAL]	*actorcap* is an inconsistent actor capability.
[K_EUNKNOWN]	*actorcap* does not specify a reachable actor.

RESTRICTIONS

For *svActorThreadList* and *svActorPortList* , the target actor and the current actor must be located on the same site.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`svSiteProbeConnect`(2MON) , `svActorProbeConnect`(2MON) , `svPortProbeConnect`(2MON) , `svThreadProbeConnect`(2MON)

NAME

svSiteMonConst – retrieve the value of site monitoring constants

SYNOPSIS

#include <mon/chMon.h>
int **svSiteMonConst**(MonSiteConst **constants*, int *size*);

FEATURES

MON

DESCRIPTION

The *svSiteMonConst* kernel call fills the *MonSiteConst* structure pointed to by the *constants* parameter. The `size` parameter must be set to *sizeof(MonSiteConst)*. It is provided for compatibility between different releases of the kernel. The members of the *MonSiteConst* structure are the following:

```
int   siteName_offset ;
int   siteName_sizeof ;
```

They specify monitoring constants related to the local site, and offsets which detail the layout of the *MonSiteState* structure (the *MonSiteState* structure gives the internal representation of the local site within the kernel data space, see *svSiteProbeConnect*(2MON)).

The *siteName_offset* member is the offset of the string representing the name of the local site. The *siteName_sizeof* member is the space reserved for the name.

The semantics of the members of the *MonSiteConst* structure is summarized by the following lines of pseudo-code:

```
MonSiteState*   siteState ;
char            (siteState + siteName_offset) [siteName_sizeof] ;
```

The special *K_ENOTIMP* value indicates that the corresponding offset or constant does not apply to this particular release of the micro-kernel.

RETURN VALUE

The return value is set to the size in bytes of the `MonSiteConst` object.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`svActorMonConst`(2MON), `svThreadMonConst`(2MON), `svPortMonConst`(2MON), `svSiteProbeConnect`(2MON)

NAME

svSiteProbeConnect, svSiteProbeDisconnect – Connect a monitoring probe to the local site; Disconnect a monitoring probe from the local site

SYNOPSIS

#include <mon/chMon.h>
int **svSiteProbeConnect**(MonSiteProbe * *probe*);

int **svSiteProbeDisconnect**(MonSiteProbe * *probe*);

FEATURES

MON

DESCRIPTION

The *svSiteProbeConnect* and *svSiteProbeDisconnect* kernel calls respectively connect a monitoring probe to, and disconnect a monitoring probe from, the local site. The *probe* parameter describes the monitoring probe. It is a pointer to a *MonSiteProbe* structure whose member is the following:

```
MonSiteVtbl*   vtbl ;
```

The *vtbl* member specifies the notification routines associated with the probe. It is a pointer to a *MonSiteVtbl* structure whose members are the following:

```
int                 vtbl_sizeof ;
Site_connection     connection ;
Site_disconnection  disconnection ;
Site_nextActor      nextActor ;
Site_intrEnter      intrEnter ;
Site_intrLeave      intrLeave ;
```

The *vtbl_sizeof* member must be set to *sizeof(MonSiteVtbl)* . It is provided for compatibility between different releases of the kernel.

The other members of the *MonSiteVtbl* structure are pointers to functions. Each member must either be null, or point to the implementation of a specific notification routine associated with the probe. If non null, the notification routine will be invoked on each occurrence of the associated event (this is described later on in this manual page). If null, no invocation is done.

The data for the *MonSiteProbe* structure, for the *MonSiteVtbl* structure, and the code for the monitoring routines must belong to supervisor space and must be locked in physical memory. The notification routines are executed in supervisor execution mode. Except where explicitly stated, they must not to re-enter the kernel except via a few permitted system calls (see *svIntrConnect(2K)).*

DESCRIPTION OF THE NOTIFICATION ROUTINES

The notification routines are the following:

```
void connection (probe, state)
   MonSiteProbe*   probe ;
   MonSiteState*   state ;
```

The *connection* routine is invoked after the probe has been connected to the local site. The *probe* parameter is a pointer to the connected probe (passed as an argument to the *svSiteProbeConnect* kernel call). The *state* parameter is a pointer to a *MonSiteState* structure, which represents the local site internally within the kernel address space (see *svSiteMonConst* (2MON)). This pointer is valid for the lifetime of the system.

```
void disconnection (probe)
   MonSiteProbe*   probe ;
```

The *disconnection* routine is invoked after the probe has been disconnected from the local site. The *probe* parameter is a pointer to the probe concerned. The rationale for this notification routine is to allow the monitoring actor to free the memory allocated for the probe safely.

```
void nextActor (probe, state)
   MonSiteProbe*   probe ;
   MonActorState*   state ;
```

The *nextActor* routine is invoked as a consequence of the *svSiteActorList* (2K) kernel call. The *probe* parameter points to the local site's probe There is one invocation of *nextActor* for each actor on the local site. The *state* parameter is a pointer to a *MonActorStat* structure which is the internal representation of the actor's state within the kernel address space. This pointer is valid for the lifetime of the actor and is guaranteed to remain valid while the notification routine is being executed. Finally, *nextActor* is invoked a last time with a null *state* parameter to indicate that all actors have been processed.

```
void intrEnter (probe, intrNo, intrCtx)
   MonSiteProbe*   probe ;
   int             intrNo ;
   void*           intrCtx ;
```

The *intrEnter* routine is invoked after the occurrence of a processor's interrupt, before that interrupt is handled by the system. The *probe* parameter is a pointer to the probe concerned. The *intrNo* parameter is the interrupt number. The *intrCtx* parameter is a pointer to the interrupt context.

The *intrNo* and *intrCtx* parameters belong to the *KnIntrSymbName* and *KnIntrCtx* types, respectively, as defined in the platform-dependent <*kbim/p_chIntr.h*> header file.

```
void intrLeave (probe, intrNo, intrCtx)
   MonSiteProbe*   probe ;
   int             intrNo ;
   void*           intrCtx ;
```

The *intrLeave* routine is invoked after the occurrence of a processor's interrupt, after that interrupt has been handled by the system. The *probe* parameter is a pointer to the probe concerned. The *intrNo* parameter is the interrupt number. The *intrCtx* parameter is a pointer to the interrupt context.

The *intrNo* and *intrCtx* parameters belong to the *KnIntrSymbName* and *KnIntrCtx* types, respectively, as defined in the platform-dependent <*kbim/p_chIntr.h*> header file.

RESTRICTIONS

The calls back *intrEnter* and *intrLeave* are not supported in this release.

RETURN VALUE

Upon successful completion, both kernel calls return 0. Otherwise, a negative error code is returned.

ERRORS

[K_EBUSY]	An attempt was made to connect a probe while another probe was already connected to the local site.
[K_EINVAL]	An attempt was made to disconnect a probe which was not connected to the local site.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`svActorProbeConnect`(2MON) , `svThreadProbeConnect`(2MON) , `svPortProbeConnect`(2MON) , `svSiteMonConst`(2MON)

NAME

svSiteProbeConnect, svSiteProbeDisconnect – Connect a monitoring probe to the local site; Disconnect a monitoring probe from the local site

SYNOPSIS

```
#include <mon/chMon.h>
int svSiteProbeConnect(MonSiteProbe * probe);

int svSiteProbeDisconnect(MonSiteProbe * probe);
```

FEATURES

MON

DESCRIPTION

The *svSiteProbeConnect* and *svSiteProbeDisconnect* kernel calls respectively connect a monitoring probe to, and disconnect a monitoring probe from, the local site. The *probe* parameter describes the monitoring probe. It is a pointer to a *MonSiteProbe* structure whose member is the following:

```
MonSiteVtbl*   vtbl ;
```

The *vtbl* member specifies the notification routines associated with the probe. It is a pointer to a *MonSiteVtbl* structure whose members are the following:

```
int                 vtbl_sizeof ;
Site_connection     connection ;
Site_disconnection  disconnection ;
Site_nextActor      nextActor ;
Site_intrEnter      intrEnter ;
Site_intrLeave      intrLeave ;
```

The *vtbl_sizeof* member must be set to *sizeof(MonSiteVtbl)* . It is provided for compatibility between different releases of the kernel.

The other members of the *MonSiteVtbl* structure are pointers to functions. Each member must either be null, or point to the implementation of a specific notification routine associated with the probe. If non null, the notification routine will be invoked on each occurrence of the associated event (this is described later on in this manual page). If null, no invocation is done.

The data for the *MonSiteProbe* structure, for the *MonSiteVtbl* structure, and the code for the monitoring routines must belong to supervisor space and must be locked in physical memory. The notification routines are executed in supervisor execution mode. Except where explicitly stated, they must not to re-enter the kernel except via a few permitted system calls (see *svIntrConnect(2K)).*

DESCRIPTION OF THE NOTIFICATION ROUTINES

The notification routines are the following:

```
void connection (probe, state)
   MonSiteProbe*   probe ;
   MonSiteState*   state ;
```

The *connection* routine is invoked after the probe has been connected to the local site. The *probe* parameter is a pointer to the connected probe (passed as an argument to the *svSiteProbeConnect* kernel call). The *state* parameter is a pointer to a *MonSiteState* structure, which represents the local site internally within the kernel address space (see *svSiteMonConst* (2MON)). This pointer is valid for the lifetime of the system.

```
void disconnection (probe)
   MonSiteProbe*   probe ;
```

The *disconnection* routine is invoked after the probe has been disconnected from the local site. The *probe* parameter is a pointer to the probe concerned. The rationale for this notification routine is to allow the monitoring actor to free the memory allocated for the probe safely.

```
void nextActor (probe, state)
   MonSiteProbe*   probe ;
   MonActorState*   state ;
```

The *nextActor* routine is invoked as a consequence of the *svSiteActorList* (2K) kernel call. The *probe* parameter points to the local site's probe There is one invocation of *nextActor* for each actor on the local site. The *state* parameter is a pointer to a *MonActorStat* structure which is the internal representation of the actor's state within the kernel address space. This pointer is valid for the lifetime of the actor and is guaranted to remain valid while the notification routine is being executed. Finally, *nextActor* is invoked a last time with a null *state* parameter to indicate that all actors have been processed.

```
void intrEnter (probe, intrNo, intrCtx)
   MonSiteProbe*   probe ;
   int             intrNo ;
   void*           intrCtx ;
```

The *intrEnter* routine is invoked after the occurrence of a processor's interrupt, before that interrupt is handled by the system. The *probe* parameter is a pointer to the probe concerned. The *intrNo* parameter is the interrupt number. The *intrCtx* parameter is a pointer to the interrupt context.

The *intrNo* and *intrCtx* parameters belong to the *KnIntrSymbName* and *KnIntrCtx* types, respectively, as defined in the platform-dependent <*kbim/p_chIntr.h*> header file.

```
void intrLeave (probe, intrNo, intrCtx)
   MonSiteProbe*   probe ;
   int             intrNo ;
   void*           intrCtx ;
```

The *intrLeave* routine is invoked after the occurrence of a processor's interrupt, after that interrupt has been handled by the system. The *probe* parameter is a pointer to the probe concerned. The *intrNo* parameter is the interrupt number. The *intrCtx* parameter is a pointer to the interrupt context.

The *intrNo* and *intrCtx* parameters belong to the *KnIntrSymbName* and *KnIntrCtx* types, respectively, as defined in the platform-dependent <*kbim/p_chIntr.h*> header file.

RESTRICTIONS

The calls back *intrEnter* and *intrLeave* are not supported in this release.

RETURN VALUE

Upon successful completion, both kernel calls return 0. Otherwise, a negative error code is returned.

ERRORS

[K_EBUSY]	An attempt was made to connect a probe while another probe was already connected to the local site.
[K_EINVAL]	An attempt was made to disconnect a probe which was not connected to the local site.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`svActorProbeConnect`(2MON) , `svThreadProbeConnect`(2MON) , `svPortProbeConnect`(2MON) , `svSiteMonConst`(2MON)

NAME

svThreadMonConst – retrieve the value of thread monitoring constants

SYNOPSIS

#include <mon/chMon.h>
int **svThreadMonConst**(MonThreadConst **constants*, int *size*);

FEATURES

MON

DESCRIPTION

The *svThreadMonConst* kernel call fills the *MonThreadConst* structure pointed to by the *constants* parameter. The `size` parameter must be set to *sizeof(MonThreadConst)*. It is provided for compatibility between different releases of the kernel. The members of the *MonThreadConst* structure are the following:

```
int   thrName_offset ;
int   thrName_sizeof ;
int   thrLid_offset ;
int   thrStatus_offset ;
int   thrStatusWaiting_mask ;
int   thrInternalVtime_offset ;
int   thrTotalVtime_offset ;
```

They specify monitoring constants related to threads, and offsets which detail the layout of the *MonThreadState* structure (the *MonThreadState* structure gives the internal representation of a thread within the kernel data space, see *svThreadProbeConnect*(2MON)).

The *thrName_offset* member is the offset of the string representing the name of the thread and the *thrName_sizeof* member is the space reserved for the name.

The *thrLid_offset* member is the offset of the local identifier of the thread.

The *thrStatus_offset* member is the offset of the "signal status". By doing a logical "and" with the signal status and the *thrStatusWaiting_mask* mask, you can find out whether the thread is waiting (out of the run queue) or not.

The *thrInternalVtime_offset* member is the offset of the KnTimeVal structure which represents the time the thread has spent running in its home actor.

The *thrTotalVtime_offset* member is the offset of the KnTimeVal structure which represents the total time the thread has spent running.

The semantics of the members of the *MonThreadConst* structure is summarized by the following lines of pseudo-code:

```
MonThreadState*  state ;
char             (state + thrName_offset) [thrName_sizeof] ;
int*             (state + thrLid_offset) ;
Bool*            (state + thrStatus) & thrStatusWaiting_mask ;

#include <util/chKnTimeVal.h>
KnTimeVal*   (state + thrInternalVtime_offset) ;
KnTimeVal*   (state + thrTotalVtime_offset) ;
```

The special *K_ENOTIMP* value indicates that the corresponding offset or constant does not apply to this particular release of the micro-kernel.

RETURN VALUE

The return value is set to the size in bytes of the `MonThreadConst` object.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`svSiteMonConst`(2MON), `svActorMonConst`(2MON), `svPortMonConst`(2MON), `svThreadProbeConnect`(2MON), `threadName`(2K)

NAME

svThreadProbeConnect, svThreadProbeDisconnect – Connect a monitoring probe to a thread; Disconnect a monitoring probe from a thread

SYNOPSIS

#include <mon/chMon.h>
int **svThreadProbeConnect**(KnCap * *actorcap*, KnThreadLid *threadli*, MonThreadProbe * *probe*);

int **svThreadProbeDisconnect**(KnCap * *actorcap*, KnThreadLid *threadli*, MonThreadProbe * *probe*);

FEATURES

MON

DESCRIPTION

The *svThreadProbeConnect* kernel call connects a monitoring probe to a thread. The *svThreadProbeDisconnect* kernel call disconnects a monitoring probe from a thread. The *threadli* parameter is the local identifier of the target thread. The *actorcap* parameter is a pointer to the capability of the target thread's actor. The *probe* parameter describes the monitoring probe. It is a pointer to a *MonThreadProbe* structure whose member is the following:

```
MonThreadVtbl*   vtbl ;
```

The *vtbl* member specifies the notification routines associated with the probe. It is a pointer to a *MonThreadVtbl* structure whose members are the following:

```
              /* House keeping */
int                   vtbl_sizeof ;
Thread_connection     connection ;
Thread_disconnection  disconnection ;
Thread_deletion       deletion ;

              /* Monitoring inheritance */
Thread_actorCreation  actorCreation ;
Thread_threadCreation threadCreation ;
Thread_portCreation   portCreation ;

              /* User specific events */
Thread_monUser        monUser ;

              /* traps and exceptions */
Thread_trapEnter      trapEnter ;
Thread_trapLeave      trapLeave ;

              /* Scheduling */
Thread_signal         signal ;
Thread_wait           wait ;
Thread_switchOn       switchOn ;
Thread_switchOff      switchOff ;
```

The *vtbl_sizeof* member must be set to *sizeof(MonThreadVtbl)* . It is provided for compatibility between different releases of the kernel.

The other members of the *MonThreadVtbl* structure are pointers to functions. Each member must either be null, or point to the implementation of a specific notification routine associated with the probe. If non null, the notification routine will be invoked on each occurrence of the associated event (this is described later on in this manual page). If null, no invocation is done.

The data for the *MonThreadProbe* structure, for the *MonThreadVtbl* structure, and the code for the monitoring routines must belong to supervisor space and must be locked in physical memory. The notification routines are executed in supervisor execution mode. Except where explicitly stated, they must not to re-enter the kernel except via a few permitted system calls (see *svIntrConnect(2K)).*

DESCRIPTION OF THE NOTIFICATION ROUTINES

The notification routines are the following:

```
void connection (probe, threadstate)
   MonThreadProbe*   probe ;
   MonThreadState*   threadstate ;
```

The *connection* routine is invoked after the probe has been connected to a thread. The *probe* parameter is a pointer to the connected probe (passed as an argument to the *svThreadProbeConnect* kernel call). The *state* parameter is a pointer to a *MonThreadState* structure, which represents the actor internally within the kernel address space (see *svMonThreadConst* (2MON)). This pointer is valid for the lifetime of the thread.

```
void disconnection (probe)
   MonThreadProbe*   probe ;
```

The *disconnection* routine is invoked after a probe has been disconnected from a thread. The *probe* parameter is a pointer to the probe concerned.

```
void deletion (probe)
   MonThreadProbe*   probe ;
```

The *deletion* routine is invoked after the actor has been deleted. The *probe* parameter is a pointer to the thread's probe.

```
MonActorProbe* actorCreation (probe, actorstate)
   MonThreadProbe*   probe ;
   MonActorState*    actorstate ;
```

The *actorCreation* routine is invoked after an actor has been created by a monitored thread (see *actorCreate* (2K)). The *probe* parameter is a pointer to the creating thread's probe. The *actorstate* parameter is a pointer to a *MonActorState* structure, which represents the created actor internally within the kernel address space (see *svMonActorConst* (2MON)). This pointer is valid for the lifetime of the actor and is guaranteed to remain valid while the notification routine is being

executed. The notification routine must return either null or a pointer to a monitoring probe for the created thread. In the latter case, the created actor will be monitored.

```
MonThreadProbe* threadCreation (probe, actorstate, threadstate, stopped)
   MonThreadProbe*   probe ;
   MonActorState*    actorstate ;
   MonThreadState*   threadstate ;
   int*              stopped ;
```

The *threadCreation* routine is invoked after a thread has been created by a monitored thread (see *threadCreate* (2K)). The *probe* parameter is a pointer to the creating thread's probe. The *threadstate* parameter is a pointer to a *MonThreadState* structure, which represents the created thread internally within the kernel address space (see *svMonThreadConst* (2MON)). This pointer is valid for the lifetime of the thread and is guaranteed to remain valid while the notification routine is being executed. The notification routine must return either null or a pointer to a monitoring probe for the created thread. In the latter case, the created thread will be monitored. The *stopped* parameter is a pointer to an integer whose value is preset to null. If the created thread should stay in the stopped state (for example in debugging situations), the integer must be set to a non-null value. In this case, the thread will return to its original created state when *threadStart* (2K) is called.

```
MonPortProbe* portCreation (probe, portstate, threadstate)
   MonThreadProbe*   probe ;
   MonPortState*     portstate ;
   MonThreadState*   threadstate ;
```

The *portCreation* routine is invoked after a port has been created by a monitored thread (see *portCreate* (2K)). The *probe* parameter is a pointer to the creating thread's probe. The *portstate* parameter is a pointer to a *MonPortState* structure, which represents the created port internally within the kernel address space (see *svMonPortConst* (2MON)). This pointer is valid for the lifetime of the port and is guaranteed to remain valid while the notification routine is being executed. The notification routine must return either null or a pointer to a monitoring probe for the created thread. In the latter case, the created port will be monitored.

```
void monUser (probe, evtno, addr, size)
   MonThreadProbe*   probe ;
   int               evtno ;
   VmAddr            addr ;
   VmSize            size ;
```

The *monUser* notification routine is invoked as a consequence of the *threadMonUser* (2MON) kernel call. The *probe* parameter is a pointer to the probe

of the thread which performed the *threadMonUser* call. The *evtno* , *addr* and `size` parameters are the same as those given as parameters to the *threadMonUser* call.

```
void trapEnter (probe, threadctx)
   MonThreadProbe*   probe ;
   KnThreadCtx*      threadctx ;
```

The *trapEnter* notification routine is invoked after a thread has performed a trap or an exception (see *svTrapConnect* (2K)), but before the trap (exception) is handled by the system. The *probe* parameter is a pointer to the probe of the thread which performed the trap. The *threadctx* parameter is a pointer to the trap context, which is a structure of the *KnThreadCtx* type. The *trapNb* member in the *KnThreadCtx* structure is the (absolute) trap number.

```
void trapLeave (probe, threadctx)
   MonThreadProbe*   probe ;
   KnThreadCtx*      threadctx ;
```

The *trapLeave* notification routine is invoked after a thread has performed a trap or an exception (see *svTrapConnect* (2K)), and after the trap (exception) has been handled by the system. The *probe* parameter is a pointer to the probe of the thread which performed the trap. The *threadctx* parameter is a pointer to the trap context, which is a structure of the *KnThreadCtx* type. The *trapNb* member in the *KnThreadCtx* structure is the (absolute) trap number.

```
void signal (probe)
   MonThreadProbe*   probe ;
```

The *signal* notification routine is invoked after a thread has been signalled by the micro-kernel executive to enter the scheduler run queue. The *probe* parameter is a pointer to the probe of the thread which was signalled.

```
void wait (probe)
   MonThreadProbe*  probe ;
```

The *wait* notification routine is invoked after a thread has opted to wait and leave the scheduler run queue. The *probe* parameter is a pointer to the probe of the thread which has opted to wait.

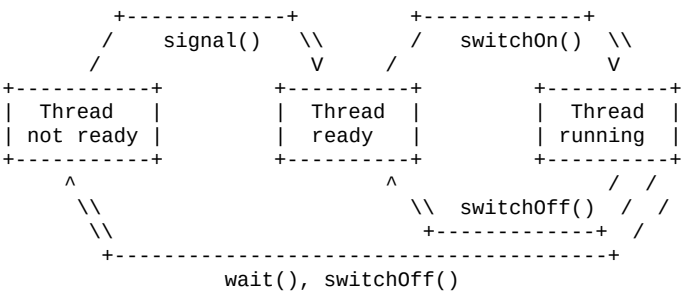
```
void switchOn (probe)
   MonThreadProbe*   probe ;
```

The *switchOn* notification routine is invoked before a thread starts running on a processor. The *probe* parameter is a pointer to the probe of the thread concerned.

```
void switchOff (probe)
   MonThreadProbe*   probe ;
```

The *switchOff* notification routine is invoked after a thread has stopped running on a processor. The *probe* parameter is a pointer to the probe of the involved thread.

The semantics of the *signal* , *wait* , *switchOn* , and *switchOff* routines are illustrated by the diagram below.



RETURN VALUE

Upon successful completion, both kernel calls return 0. Otherwise, a negative error code is returned.

ERRORS

[K_EBUSY]	An attempt was made to connect a probe while another probe was already connected to the thread.
[K_EINVAL]	An attempt was made to disconnect a probe which was not connected to the thread.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`svSiteProbeConnect`(2MON) , `svActorProbeConnect`(2MON) , `svPortProbeConnect`(2MON) , `svMonThreadConst`(2MON)

NAME

svThreadProbeConnect, svThreadProbeDisconnect – Connect a monitoring probe to a thread; Disconnect a monitoring probe from a thread

SYNOPSIS

#include <mon/chMon.h>
int **svThreadProbeConnect**(KnCap * *actorcap*, KnThreadLid *threadli*, MonThreadProbe * *probe*);

int **svThreadProbeDisconnect**(KnCap * *actorcap*, KnThreadLid *threadli*, MonThreadProbe * *probe*);

FEATURES

MON

DESCRIPTION

The *svThreadProbeConnect* kernel call connects a monitoring probe to a thread. The *svThreadProbeDisconnect* kernel call disconnects a monitoring probe from a thread. The *threadli* parameter is the local identifier of the target thread. The *actorcap* parameter is a pointer to the capability of the target thread's actor. The *probe* parameter describes the monitoring probe. It is a pointer to a *MonThreadProbe* structure whose member is the following:

```
MonThreadVtbl*   vtbl ;
```

The *vtbl* member specifies the notification routines associated with the probe. It is a pointer to a *MonThreadVtbl* structure whose members are the following:

```
              /* House keeping */
int                     vtbl_sizeof ;
Thread_connection       connection ;
Thread_disconnection    disconnection ;
Thread_deletion         deletion ;

              /* Monitoring inheritance */
Thread_actorCreation    actorCreation ;
Thread_threadCreation   threadCreation ;
Thread_portCreation     portCreation ;

              /* User specific events */
Thread_monUser          monUser ;

              /* traps and exceptions */
Thread_trapEnter        trapEnter ;
Thread_trapLeave        trapLeave ;

              /* Scheduling */
Thread_signal           signal ;
Thread_wait             wait ;
Thread_switchOn         switchOn ;
Thread_switchOff        switchOff ;
```

The *vtbl_sizeof* member must be set to *sizeof(MonThreadVtbl)* . It is provided for compatibility between different releases of the kernel.

The other members of the *MonThreadVtbl* structure are pointers to functions. Each member must either be null, or point to the implementation of a specific notification routine associated with the probe. If non null, the notification routine will be invoked on each occurrence of the associated event (this is described later on in this manual page). If null, no invocation is done.

The data for the *MonThreadProbe* structure, for the *MonThreadVtbl* structure, and the code for the monitoring routines must belong to supervisor space and must be locked in physical memory. The notification routines are executed in supervisor execution mode. Except where explicitly stated, they must not to re-enter the kernel except via a few permitted system calls (see *svIntrConnect(2K)).*

DESCRIPTION OF THE NOTIFICATION ROUTINES

The notification routines are the following:

```
void connection (probe, threadstate)
   MonThreadProbe*   probe ;
   MonThreadState*   threadstate ;
```

The *connection* routine is invoked after the probe has been connected to a thread. The *probe* parameter is a pointer to the connected probe (passed as an argument to the *svThreadProbeConnect* kernel call). The *state* parameter is a pointer to a *MonThreadState* structure, which represents the actor internally within the kernel address space (see *svMonThreadConst* (2MON)). This pointer is valid for the lifetime of the thread.

```
void disconnection (probe)
   MonThreadProbe*   probe ;
```

The *disconnection* routine is invoked after a probe has been disconnected from a thread. The *probe* parameter is a pointer to the probe concerned.

```
void deletion (probe)
   MonThreadProbe*   probe ;
```

The *deletion* routine is invoked after the actor has been deleted. The *probe* parameter is a pointer to the thread's probe.

```
MonActorProbe* actorCreation (probe, actorstate)
   MonThreadProbe*   probe ;
   MonActorState*    actorstate ;
```

The *actorCreation* routine is invoked after an actor has been created by a monitored thread (see *actorCreate* (2K)). The *probe* parameter is a pointer to the creating thread's probe. The *actorstate* parameter is a pointer to a *MonActorState* structure, which represents the created actor internally within the kernel address space (see *svMonActorConst* (2MON)). This pointer is valid for the lifetime of the actor and is guaranteed to remain valid while the notification routine is being

executed. The notification routine must return either null or a pointer to a monitoring probe for the created thread. In the latter case, the created actor will be monitored.

```
MonThreadProbe* threadCreation (probe, actorstate, threadstate, stopped)
   MonThreadProbe*   probe ;
   MonActorState*    actorstate ;
   MonThreadState*   threadstate ;
   int*              stopped ;
```

The *threadCreation* routine is invoked after a thread has been created by a monitored thread (see *threadCreate* (2K)). The *probe* parameter is a pointer to the creating thread's probe. The *threadstate* parameter is a pointer to a *MonThreadState* structure, which represents the created thread internally within the kernel address space (see *svMonThreadConst* (2MON)). This pointer is valid for the lifetime of the thread and is guaranteed to remain valid while the notification routine is being executed. The notification routine must return either null or a pointer to a monitoring probe for the created thread. In the latter case, the created thread will be monitored. The *stopped* parameter is a pointer to an integer whose value is preset to null. If the created thread should stay in the stopped state (for example in debugging situations), the integer must be set to a non-null value. In this case, the thread will return to its original created state when *threadStart* (2K) is called.

```
MonPortProbe* portCreation (probe, portstate, threadstate)
   MonThreadProbe*   probe ;
   MonPortState*     portstate ;
   MonThreadState*   threadstate ;
```

The *portCreation* routine is invoked after a port has been created by a monitored thread (see *portCreate* (2K)). The *probe* parameter is a pointer to the creating thread's probe. The *portstate* parameter is a pointer to a *MonPortState* structure, which represents the created port internally within the kernel address space (see *svMonPortConst* (2MON)). This pointer is valid for the lifetime of the port and is guaranteed to remain valid while the notification routine is being executed. The notification routine must return either null or a pointer to a monitoring probe for the created thread. In the latter case, the created port will be monitored.

```
void monUser (probe, evtno, addr, size)
   MonThreadProbe*   probe ;
   int               evtno ;
   VmAddr            addr ;
   VmSize            size ;
```

The *monUser* notification routine is invoked as a consequence of the *threadMonUser* (2MON) kernel call. The *probe* parameter is a pointer to the probe

of the thread which performed the *threadMonUser* call. The *evtno* , *addr* and `size` parameters are the same as those given as parameters to the *threadMonUser* call.

```
void trapEnter (probe, threadctx)
   MonThreadProbe*   probe ;
   KnThreadCtx*      threadctx ;
```

The *trapEnter* notification routine is invoked after a thread has performed a trap or an exception (see *svTrapConnect* (2K)), but before the trap (exception) is handled by the system. The *probe* parameter is a pointer to the probe of the thread which performed the trap. The *threadctx* parameter is a pointer to the trap context, which is a structure of the *KnThreadCtx* type. The *trapNb* member in the *KnThreadCtx* structure is the (absolute) trap number.

```
void trapLeave (probe, threadctx)
   MonThreadProbe*   probe ;
   KnThreadCtx*      threadctx ;
```

The *trapLeave* notification routine is invoked after a thread has performed a trap or an exception (see *svTrapConnect* (2K)), and after the trap (exception) has been handled by the system. The *probe* parameter is a pointer to the probe of the thread which performed the trap. The *threadctx* parameter is a pointer to the trap context, which is a structure of the *KnThreadCtx* type. The *trapNb* member in the *KnThreadCtx* structure is the (absolute) trap number.

```
void signal (probe)
   MonThreadProbe*   probe ;
```

The *signal* notification routine is invoked after a thread has been signalled by the micro-kernel executive to enter the scheduler run queue. The *probe* parameter is a pointer to the probe of the thread which was signalled.

```
void wait (probe)
   MonThreadProbe*  probe ;
```

The *wait* notification routine is invoked after a thread has opted to wait and leave the scheduler run queue. The *probe* parameter is a pointer to the probe of the thread which has opted to wait.

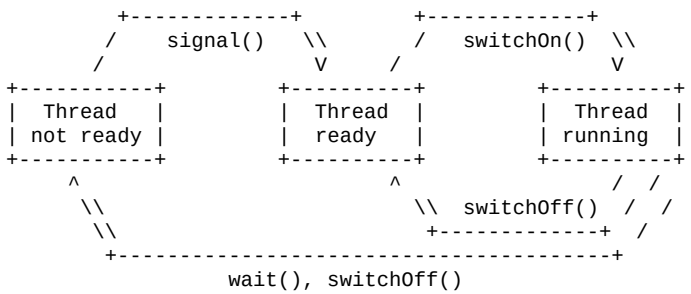
```
void switchOn (probe)
   MonThreadProbe*   probe ;
```

The *switchOn* notification routine is invoked before a thread starts running on a processor. The *probe* parameter is a pointer to the probe of the thread concerned.

```
void switchOff (probe)
   MonThreadProbe*   probe ;
```

The *switchOff* notification routine is invoked after a thread has stopped running on a processor. The *probe* parameter is a pointer to the probe of the involved thread.

The semantics of the *signal* , *wait* , *switchOn* , and *switchOff* routines are illustrated by the diagram below.



RETURN VALUE

Upon successful completion, both kernel calls return 0. Otherwise, a negative error code is returned.

ERRORS

[K_EBUSY]	An attempt was made to connect a probe while another probe was already connected to the thread.
[K_EINVAL]	An attempt was made to disconnect a probe which was not connected to the thread.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`svSiteProbeConnect`(2MON) , `svActorProbeConnect`(2MON) , `svPortProbeConnect`(2MON) , `svMonThreadConst`(2MON)

NAME

threadMonUser – generate a user-specific monitoring event

SYNOPSIS

```
#include <mon/chMon.h>
int threadMonUser(int evtNo, VmAddr dataAddr, VmSize dataSize);
```

FEATURES

MON

DESCRIPTION

The *threadMonUser* system call generates a user-specific monitoring event (see the *monUser* notification routine on the *svThreadProbeConnect*(2MON) manual page).

The *evtNo* parameter is the number of the user-specific event. The *dataAddr* parameter is the address of the user-provided data for the event. The *dataSize* parameter is the size of the user-provided data. These three parameters are opaque to the kernel and are passed "as is" to the *monUser* monitoring notification routine.

RETURN VALUE

The *threadMonUser* system call returns 0.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

svThreadProbeConnect(2MON)

NAME

KcModule, UcModule – Array of all micro-kernel supervisor modules and system calls; Array of all micro-kernel user modules and system calls

SYNOPSIS

```
#include <mon/chModules.h>
extern KernelModule KcModule [K_CMODULE_MAX];
extern KernelModule UcModule [K_CMODULE_MAX];
```

FEATURES

MON

DESCRIPTION

The `KcModule` array contains a description of the supervisor modules and system calls available on the system. Each entry in the array is a *KernelModule* structure whose members are the following:

```
char*         kmName ;
int           kmScCnt ;
KernelCall*   kmSc ;
```

The *kmName* member specifies the name of the module. The *kmScCnt* member specifies the number of system calls defined by the module. The *kmSc* member is a pointer to an array describing the module's (supervisor) system calls. This array has *kmScCnt* entries; the n-th entry contains the description of the n-th system call of the module. The entries are of the type *KernelCall* . The *KernelCall* structure has the following members:

```
char*   kcName ;
int     kcModule ;
int     kcNo ;
int     kcArgsCnt ;
```

The *kcName* member specifies the name of the system call. The *kcModule* member specifies the module to which the system call belongs. The *kcNo* member specifies the system call's number. The *kcArgsCnt* member specifies the number of arguments used by the system call.

The `<mon/chKcalls.h>` header file defines a macro for each supervisor system call, enabling the system call to be mapped to the corresponding entry in the *KcModule* array. For example:

```
#define svTrapConnect_sc (&KcModule[0].kmSc[24])
```

The `UcModule` array contains a description of the user modules and system calls available on the system. Its layout is the same as the *KcModule* array.

The `<mon/chUcalls.h>` header file defines a macro for each user system call, enabling the system call to be mapped to the corresponding entry in the *UcModule* array. For example:

```
#define actorCreate_uc (&UcModule[0].kmSc[1])
```

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

Index

T

U