Kernel

Reference Manual
Copyright

Copyright (C) 2005 by Litronic AG. All rights reserved. No part of this publication may be reproduced, transmitted, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, optical, chemical or otherwise, without the prior written permission of Litronic AG. The Software described in this document is licensed under a software license agreement and maybe used only in accordance with the terms of this agreement.

Disclaimer

Litronic AG, makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability of fitness for any particular purpose. Further, Litronic AG, reserves the right to revise this publication and to make changes from time to time in the contents hereof without obligation to Litronic AG to notify any person of such revision or changes.

Trademark

SCIOPTA is a registered trademark of Litronic AG.

Headquarters
Litronic AG
Gartenstrasse 76
CH-4052 Basel
Switzerland
Tel. +41 61 276 90 90
Fax +41 61 276 90 99
email: sales@sciopta.com
www.sciopta.com

Germany
Sciopta Systems GmbH
Hauptstrasse 293
D-79576 Weil am Rhein
Germany
Tel. +49 7621 940 919 0
Fax +49 7621 940 919 19
email: sales@sciopta.com
www.sciopta.com

France
Sciopta Systems France
3, boulevard de l’Europe
F-68100 Mulhouse
France
Tel. +33 3 89 36 33 91
Fax +33 3 89 45 57 10
email: sales@sciopta.com
www.sciopta.com
# Table of Contents

1. **Introduction** .................................................................................................................. 1-1
   1.1 SCIOPTA Real-Time Operating System ............................................................................ 1-1
   1.2 About This Manual ............................................................................................................ 1-1
   1.3 Real-Time Operating System Overview ........................................................................... 1-2
   1.3.1 Management Duties ......................................................................................................... 1-2
   1.3.1.1 CPU Management ...................................................................................................... 1-3
   1.3.1.2 Memory Management ................................................................................................. 1-3
   1.3.1.3 Input/Output Management .......................................................................................... 1-3
   1.3.1.4 Time Management ..................................................................................................... 1-3
   1.3.1.5 Interprocess Communication ...................................................................................... 1-3

2. **System Call Summary** .................................................................................................. 2-1
   2.1 System Calls Overview listed in Functional Groups ......................................................... 2-1
   2.1.1 Message System Calls ..................................................................................................... 2-1
   2.1.2 Process System Calls ....................................................................................................... 2-2
   2.1.3 Module System Calls ...................................................................................................... 2-3
   2.1.4 Message Pool Calls ........................................................................................................ 2-4
   2.1.5 Timing Calls .................................................................................................................... 2-5
   2.1.6 System Tick Calls ........................................................................................................... 2-5
   2.1.7 Process Trigger Calls ...................................................................................................... 2-5
   2.1.8 Process Variable Calls ................................................................................................... 2-6
   2.1.9 Connector Process Calls .................................................................................................. 2-6
   2.1.10 Miscellaneous and Error Calls ..................................................................................... 2-6

3. **System Call Reference** .................................................................................................. 3-1
   3.1 sc_connectorRegister .......................................................................................................... 3-1
   3.2 sc_connectorUnregister ....................................................................................................... 3-2
   3.3 sc_mscCrc .......................................................................................................................... 3-3
   3.4 sc_mscCrcContd .................................................................................................................. 3-4
   3.5 sc_mscErrnoGet ................................................................................................................. 3-5
   3.6 sc_mscErrnoSet .................................................................................................................... 3-6
   3.7 sc_mscError ....................................................................................................................... 3-7
   3.8 sc_mscErrorHookRegister ................................................................................................. 3-8
   3.9 sc_moduleCreate ................................................................................................................. 3-9
   3.10 sc_moduleFriendAdd ....................................................................................................... 3-12
   3.11 sc_moduleFriendAll ......................................................................................................... 3-13
   3.12 sc_moduleFriendGet ......................................................................................................... 3-14
   3.13 sc_moduleFriendNone ..................................................................................................... 3-15
   3.14 sc_moduleFriendRm ...................................................................................................... 3-16
   3.15 sc_moduleIdGet .............................................................................................................. 3-17
   3.16 sc_moduleInfo ................................................................................................................... 3-18
   3.17 sc_moduleKill .................................................................................................................... 3-19
   3.18 sc_moduleNameGet ......................................................................................................... 3-20
   3.19 sc_msgAcquire ................................................................................................................. 3-21
   3.20 sc_msgAddrGet ............................................................................................................... 3-22
   3.21 sc_msgAlloc ...................................................................................................................... 3-23
   3.22 sc_msgAllocClr ............................................................................................................... 3-25
   3.23 sc_msgFree ..................................................................................................................... 3-26
   3.24 sc_msgHookRegister ....................................................................................................... 3-27
| Reference Manual |
|------------------|-----------------|
| SCIOPTA - Kernel | Manual Version 1.6 |

3.25 sc_msgOwnerGet .......................................................... 3-28
3.26 sc_msgPoolIdGet .......................................................... 3-29
3.27 sc_msgRx .................................................................. 3-30
3.28 sc_msgSndGet .............................................................. 3-32
3.29 sc_msgSizeGet ............................................................. 3-33
3.30 sc_msgSizeSet .............................................................. 3-34
3.31 sc_msgTx .................................................................. 3-35
3.32 sc_msgTxAlias ............................................................... 3-36
3.33 sc_poolCreate ................................................................. 3-37
3.34 sc_poolDefault .............................................................. 3-39
3.35 sc_poolHookRegister ......................................................... 3-40
3.36 sc_poolIdGet ................................................................. 3-41
3.37 sc_poolInfo ................................................................ 3-42
3.38 sc_poolKill ................................................................. 3-43
3.39 sc_poolReset ............................................................... 3-44
3.40 sc_procCreate ............................................................... 3-45
3.41 sc_procDaemonRegister .................................................... 3-48
3.42 sc_procDaemonUnregister .................................................. 3-49
3.43 sc_procHookRegister ....................................................... 3-50
3.44 sc_procIdGet ................................................................. 3-51
3.45 sc_procIntCreate ............................................................ 3-53
3.46 sc_procKill ................................................................ 3-55
3.47 sc_procNameCreate .......................................................... 3-56
3.48 sc_procObserver .............................................................. 3-57
3.49 sc_procPathGet .............................................................. 3-58
3.50 sc_procPpidGet .............................................................. 3-59
3.51 sc_procPrioCreate ........................................................... 3-60
3.52 sc_procPrioGet ............................................................... 3-62
3.53 sc_procPrioSet ............................................................... 3-63
3.54 sc_procSchedLock ............................................................ 3-64
3.55 sc_procSchedUnlock ......................................................... 3-65
3.56 sc_procSliceGet ............................................................. 3-66
3.57 sc_procSliceSet ............................................................. 3-67
3.58 sc_procStart ................................................................. 3-68
3.59 sc_procStop ................................................................ 3-69
3.60 sc_procTimCreate ........................................................... 3-70
3.61 sc_procUnobserver .......................................................... 3-72
3.62 sc_procUsrIntCreate ........................................................ 3-73
3.63 sc_procVarDel ............................................................... 3-75
3.64 sc_procVarGet ............................................................... 3-76
3.65 sc_procVarInit ............................................................... 3-77
3.66 sc_procVarRm ............................................................... 3-78
3.67 sc_procVarSet ............................................................... 3-79
3.68 sc_procVectorGet ............................................................ 3-80
3.69 sc_procYield ................................................................. 3-81
3.70 sc_sleep .................................................................... 3-82
3.71 sc_tick .................................................................... 3-83
3.72 sc_tickGet .................................................................. 3-84
3.73 sc_tickLength ............................................................... 3-85
3.74 sc_tickMs2Tick .............................................................. 3-86
3.75 sc_tickTick2Ms ............................................................. 3-87
3.76 sc_tmoAdd ................................................................. 3-88
Table of Contents

3.77 sc_tmoRm .......................................................... 3-89
3.78 sc_trigger .......................................................... 3-90
3.79 sc_triggerValueGet .................................................. 3-91
3.80 sc_triggerValueSet .................................................. 3-92
3.81 sc_triggerWait .................................................. 3-93

4. Kernel Error Codes .................................................. 4-1
4.1 Introduction .......................................................... 4-1
4.2 Include Files .......................................................... 4-1
4.3 Function Codes .................................................. 4-2
4.4 Error Codes .................................................. 4-4
4.5 Error Types .................................................. 4-5

5. Manual Revision .................................................. 5-1
5.1 Manual Version 1.6 .................................................. 5-1
5.2 Manual Version 1.5 .................................................. 5-1
5.3 Manual Version 1.4 .................................................. 5-1
5.4 Manual Version 1.3 .................................................. 5-2
5.5 Manual Version 1.2 .................................................. 5-3
5.6 Manual Version 1.1 .................................................. 5-4
5.7 Manual Version 1.0 .................................................. 5-4

6. Index .......................................................... 6-1
1 Introduction

1.1 SCIOPTA Real-Time Operating System

SCIOPTA is a high-performance real-time operating system for a variety of target processors. The operating system environment includes:

- Pre-emptive multi-tasking real-time kernel
- BSP - Board support packages
- IPS - Internet protocols
- SFS - File system
- CONNECTOR - support for distributed multi-CPU systems
- SMMS - support for Memory Management Units and system protection
- DRUID - system level debug suite

1.2 About This Manual

The purpose of this SCIOPTA - Kernel, Reference Manual is to give all needed information how to use the SCIOPTA real-time kernel in an embedded project.

This reference part contains a complete description of all system calls and error messages.

Please consult the SCIOPTA - Kernel, User’s Guide for detailed information about the technologies and methods used in the SCIOPTA kernels and useful information about system design and configuration.

The manual includes all information which are not target specific and valid for all supported CPUs. The target specific information can be found in the SCIOPTA - Target Manual which is different for each SCIOPTA supported processor family and includes:

- Installation Information
- Getting started examples
- Description of the system configuration (SCONF tool)
- Information about the system building procedures
- Description of the board support packages (BSP)
- List of distributed files
- Release notes and version history
1 Introduction

1.3 Real-Time Operating System Overview

A real-time operating system (RTOS) is the core control software in a real-time system.

In a real-time system it must be guaranteed that specific tasks respond to external events within a limited and specified time.

Figure 1-1 shows a typical part of a real-time system. An external interrupt is activating an interrupt process which allocates a message and transmits the message to a prioritized process. The time \( t \) between the occurrence of the interrupt and the processing of the interrupt in process B must not exceed a specified maximum time under any circumstances. This maximum time must not depend on system resources such as number of processes or number of messages.

1.3.1 Management Duties

A real-time operating system fulfils many different tasks such as:

- resource management (CPU, Memory and I/O)
- time management
- interprocess communication management
1 Introduction

1.3.1.1 CPU Management

As a user of a real-time operating system you will divide your program into a number of small program parts. These parts are called processes and will normally operate independently of each other and be connected through some interprocess communication connections.

It is obvious that only one process can use the CPU at a time. One important task of the real-time operating system is to activate the various processes according to their importance. The user can control this by assigning priorities to the processes.

The real-time operating system guarantees the execution of the most important part of a program at any particular moment.

1.3.1.2 Memory Management

The real-time operating system will control the memory needs and accesses of a system and always guarantee the real-time behaviour of the system. Specific functions and techniques are offered by a real-time operating system to protect memory from writing by processes that should have no access to them.

Thus, allocating, freeing and protecting of memory buffers used by processes are one of the main duties of the memory management part of a real-time operating system.

1.3.1.3 Input/Output Management

Another important task of a real-time operating system is to support the user in designing the interfaces for various hardware such as input/output ports, displays, communication equipment, storage devices etc.

1.3.1.4 Time Management

In a real-time system it is very important to manage time-dependent applications and functions appropriately. There are many timing demands in a real-time system such as notifying the user after a certain time, activating particular tasks cyclically or running a function for a specified time. A real-time operating system must be able to manage these timing requirements by scheduling activities at, or after a certain specified time.

1.3.1.5 Interprocess Communication

The designer of a real-time system will divide the whole system into processes. One design goal of a real-time system is to keep the processes as isolated as possible. Even so, it is often necessary to exchange data between processes.

Interprocess relations can occur in many different forms such as global variables, function calls, timing interactions, priority relationships, interrupt enabling/disabling, semaphore, message passing.

One of the duties of a real-time operating system is to manage interprocess communication and to control exchange of data between processes.
2 System Call Summary

2.1 System Calls Overview listed in Functional Groups

2.1.1 Message System Calls

- **sc_msgAcquire**: Changes the owner of the message. The caller becomes the owner of the message. Chapter 3.19, page 3-21.
- **sc_msgAddrGet**: Returns the process ID of the addressee of the message. Chapter 3.20, page 3-22.
- **sc_msgAlloc**: Allocates a memory buffer of selectable size from a message pool. Chapter 3.21, page 3-23.
- **sc_msgAllocClr**: Allocates a memory buffer of selectable size from a message pool and initializes the message data to 0. Chapter 3.22, page 3-25.
- **sc_msgFree**: Returns a message to the message pool. Chapter 3.23, page 3-26.
- **sc_msgHookRegister**: Registers a message hook. Chapter 3.24, page 3-27.
- **sc_msgOwnerGet**: Returns the process ID of the owner of the message. Chapter 3.25, page 3-28.
- **sc_msgPoolIdGet**: Returns the pool ID of a message. Chapter 3.26, page 3-29.
- **sc_msgRx**: Receives one or more defined messages. Chapter 3.27, page 3-30.
- **sc_msgSndGet**: Returns the process ID of the sender of the message. Chapter 3.28, page 3-32.
- **sc_msgSizeGet**: Returns the size of the message buffer. Chapter 3.29, page 3-33.
- **sc_msgSizeSet**: Modifies the size of a message buffer. Chapter 3.30, page 3-34.
- **sc_msgTx**: Sends a message to a process. Chapter 3.31, page 3-35.
- **sc_msgTxAlias**: Sends a message to a process by setting any process ID. Chapter 3.32, page 3-36.
2 System Call Summary

2.1.2 Process System Calls

sc_procCreate
Requests the kernel daemon to create process.
Chapter 3.40, page 3-45. This system call is only available by the SCIOPTA Compact kernel.

sc_procDaemonRegister
Registers a process daemon which is responsible for pidGet request.
Chapter 3.41, page 3-48. This system call is not supported by the SCIOPTA Compact kernel.

sc_procDaemonUnregister
Unregisters a process daemon.
Chapter 3.42, page 3-49. This system call is not supported by the SCIOPTA Compact kernel.

sc_procHookRegister
Registers a process hook.
Chapter 3.43, page 3-50.

sc_procIdGet
Returns the process ID of a process.
Chapter 3.44, page 3-51.

sc_procIntCreate
Requests the kernel daemon to create an interrupt process.
Chapter 3.45, page 3-53. This system call is not supported by the SCIOPTA Compact kernel.

sc_procKill

sc_procNameGet
Returns the full name of a process.
Chapter 3.47, page 3-56.

sc_procObserve
Request a message to be sent if the given process pid dies (process supervision).
Chapter 3.48, page 3-57. This system call is not supported by the SCIOPTA Compact kernel.

sc_procPathGet
Returns the path of a process.
Chapter 3.49, page 3-58.

sc_procPpidGet
Returns the process ID of the parent of a process.
Chapter 3.50, page 3-59. This system call is not supported by the SCIOPTA Compact kernel.

sc_procPrioCreate
Requests the kernel daemon to create a prioritized process.
Chapter 3.51, page 3-60. This system call is not supported by the SCIOPTA Compact kernel.

sc_procPrioGet
Returns the priority of a process.
Chapter 3.52, page 3-62.

sc_procPrioSet
Sets the priority of a process.
Chapter 3.53, page 3-63.

sc_procSchedLock
Locks the scheduler and returns the number of times it has been locked before.
Chapter 3.54, page 3-64.

sc_procSchedUnlock
Unlocks the scheduler by decrementing the lock counter by one.
Chapter 3.55, page 3-65.
2 System Call Summary

sc_procsliceGet
Returns the time slice of a timer process.
Chapter 3.56, page 3-66.

sc_procsliceSet
Sets the time slice of a timer process.
Chapter 3.57, page 3-67.

sc_procStart
Starts a process.
Chapter 3.58, page 3-68.

sc_procStop
Stops a process.
Chapter 3.59, page 3-69.

sc_proctimeCreate
Requests the kernel daemon to create a timer process.
Chapter 3.60, page 3-70. This system call is not supported by the SCIOPTA Compact kernel.

sc_procUnobserve
Cancels the observation of a process.
Chapter 3.61, page 3-72. This system call is not supported by the SCIOPTA Compact kernel.

sc_procusrIntCreate
Requests the kernel daemon to create a user interrupt process.
Chapter 3.62, page 3-73. This system call is not supported by the SCIOPTA Compact kernel.

sc_procvectorGet
Returns the interrupt vector of an interrupt process.
Chapter 3.68, page 3-80.

sc_procYield
Yields the CPU to the next ready process within the current process's priority group.
Chapter 3.69, page 3-81.

2.1.3 Module System Calls

Please Note:
The module concept is not supported in the SCIOPTA Compact Kernel. All system calls described in this chapter 2.1.3 “Module System Calls” are not available in the SCIOPTA Compact Kernel. The SCIOPTA Compact Kernel has only one module (the system module).

sc_moduleCreate
Creates a module.
Chapter 3.9, page 3-9. This system call is not supported by the SCIOPTA Compact kernel.

sc_moduleFriendAdd
Adds a module to the module friend set.
Chapter 3.10, page 3-12. This system call is not supported by the SCIOPTA Compact kernel.

sc_moduleFriendAll
Sets a modules as friend.
Chapter 3.11, page 3-13. This system call is not supported by the SCIOPTA Compact kernel.
2 System Call Summary

sc_moduleFriendGet
Returns if module is member of a module friend set.
Chapter 3.12, page 3-14. This system call is not supported by the SCIOPTA Compact kernel.

sc_moduleFriendNone
Removes all modules from a module friend set.
Chapter 3.13, page 3-15. This system call is not supported by the SCIOPTA Compact kernel.

sc_moduleFriendRm
Removes a module from the module friend set.
Chapter 3.14, page 3-16. This system call is not supported by the SCIOPTA Compact kernel.

sc_moduleIdGet
Returns the ID of a module
Chapter 3.15, page 3-17. This system call is not supported by the SCIOPTA Compact kernel.

sc_moduleInfo
Returns a snap-shot of a module control block.
Chapter 3.16, page 3-18. This system call is not supported by the SCIOPTA Compact kernel.

sc_moduleKill
Kills a module.
Chapter 3.17, page 3-19. This system call is not supported by the SCIOPTA Compact kernel.

sc_moduleNameGet
Returns the full name of a module.
Chapter 3.18, page 3-20. This system call is not supported by the SCIOPTA Compact kernel.

2.1.4 Message Pool Calls

sc_poolCreate
Creates a message pool.
Chapter 3.33, page 3-37.

sc_poolDefault
Sets a message pool as default pool.
Chapter 3.34, page 3-39.

sc_poolHookRegister
Registers a pool hook.
Chapter 3.35, page 3-40.

sc_poolIdGet
Returns the ID of a message pool.
Chapter 3.36, page 3-41.

sc_poolInfo
Returns a snap-shot of a pool control block.
Chapter 3.37, page 3-42.

sc_poolKill
Kills a message pool.
Chapter 3.38, page 3-43.

sc_poolReset
Resets a message pool in its original state.
Chapter 3.39, page 3-44.
2 System Call Summary

2.1.5 Timing Calls

**sc_sleep**
Suspends a process for a defined time.
Chapter 3.70, page 3-82.

**sc_tmoAdd**
Request a time-out message after a defined time.
Chapter 3.76, page 3-88.

**sc_tmoRm**
Remove a time-out job.
Chapter 3.77, page 3-89.

2.1.6 System Tick Calls

**sc_tick**
Calls the kernel tick function. Advances the kernel tick counter by 1.
Chapter 3.71, page 3-83.

**sc_tickGet**
Returns the actual kernel tick counter value.
Chapter 3.72, page 3-84.

**sc_tickLength**
Returns/sets the current system tick-length.
Chapter 3.73, page 3-85.

**sc_tickMs2Tick**
Converts a time from milliseconds into ticks.
Chapter 3.74, page 3-86.

**sc_tickTick2Ms**
Converts a time from ticks into milliseconds.
Chapter 3.75, page 3-87.

2.1.7 Process Trigger Calls

**sc_trigger**
Signals a process trigger.
Chapter 3.78, page 3-90.

**sc_triggerValueGet**
Returns the value of a process trigger.
Chapter 3.79, page 3-91.

**sc_triggerValueSet**
Sets the value of a process trigger.
Chapter 3.80, page 3-92.

**sc_triggerWait**
Waits on its process trigger.
Chapter 3.81, page 3-93.
2 System Call Summary

2.1.8 Process Variable Calls

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Description</th>
<th>Chapter</th>
</tr>
</thead>
<tbody>
<tr>
<td>sc_procVarDel</td>
<td>Deletes a process variable.</td>
<td>Chapter 3.63, page 3-75.</td>
</tr>
<tr>
<td>sc_procVarGet</td>
<td>Returns a process variable.</td>
<td>Chapter 3.64, page 3-76.</td>
</tr>
<tr>
<td>sc_procVarInit</td>
<td>Initializes a process variable area.</td>
<td>Chapter 3.65, page 3-77.</td>
</tr>
<tr>
<td>sc_procVarRm</td>
<td>Removes a process variable area.</td>
<td>Chapter 3.66, page 3-78.</td>
</tr>
<tr>
<td>sc_procVarSet</td>
<td>Sets a process variable.</td>
<td>Chapter 3.67, page 3-79.</td>
</tr>
</tbody>
</table>

2.1.9 Connector Process Calls

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Description</th>
<th>Chapter</th>
</tr>
</thead>
<tbody>
<tr>
<td>sc_connectorRegister</td>
<td>Registers a connector process.</td>
<td>Chapter 3.1, page 3-1.</td>
</tr>
<tr>
<td>sc_connectorUnregister</td>
<td>Removes a registered connector process.</td>
<td>Chapter 3.2, page 3-2.</td>
</tr>
</tbody>
</table>

2.1.10 Miscellaneous and Error Calls

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Description</th>
<th>Chapter</th>
</tr>
</thead>
<tbody>
<tr>
<td>sc_miscCrc</td>
<td>Calculates a CRC over a specified memory range.</td>
<td>Chapter 3.3, page 3-3. <strong>This system call is not supported by the SCIOPTA Compact kernel.</strong></td>
</tr>
<tr>
<td>sc_miscCrcContd</td>
<td>Calculates a CRC over an additional memory range.</td>
<td>Chapter 3.4, page 3-4. <strong>This system call is not supported by the SCIOPTA Compact kernel.</strong></td>
</tr>
<tr>
<td>sc_miscErrnoSet</td>
<td>Sets the error code.</td>
<td>Chapter 3.6, page 3-6.</td>
</tr>
<tr>
<td>sc_miscErrnoGet</td>
<td>Returns the error code.</td>
<td>Chapter 3.5, page 3-5.</td>
</tr>
<tr>
<td>sc_miscErrorHookRegister</td>
<td>Registers an Error Hook.</td>
<td>Chapter 3.8, page 3-8.</td>
</tr>
</tbody>
</table>
3 System Call Reference

3.1 sc_connectorRegister

This system call is used to register a connector process. The caller becomes a connector process.

Connector processes are used to connect different target in distributed SCIOPTA systems. Messages sent to external processes (residing on remote target or CPU) are sent to the local connector processes.

Please consult the SCIOPTA - Kernel, User’s Guide for more information about CONNECTOR processes.

```
s_c_pid_t sc_connectorRegister {
    int defaultConn
}
```

Parameter

defaultConn

This parameter can be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The caller becomes a connector process. The name of the process correspond to the name of the target.</td>
</tr>
<tr>
<td>!=0</td>
<td>The caller becomes the default connector for the system.</td>
</tr>
</tbody>
</table>

Return Value

Specific connector ID which is used to define the process ID for distributed processes.
3.2 sc_connectorUnregister

This system call is used to remove a registered connector process. The caller becomes a normal prioritized process.
Please consult SCIOPTA - Kernel, User’s for more information about CONNECTOR processes.

```c
void sc_connectorUnregister (void);
```

Parameter

none

Return Value

none
3  System Call Reference

3.3  sc_miscCrc

Please Note: The system call sc_miscCrc is not supported in the SCIOPTA Compact Kernel.

This function calculates a 16 bit CRC over a specified memory range.
The start value of the CRC is 0xFFFF.

```c
__u16 sc_miscCrc ( __u8 *data, unsigned int len );
```

**Parameter**

*data*
  Pointer to the memory range.

*len*
  Number of bytes

**Return Value**

The 16 bit CRC value.
### 3.4 sc_miscCrcContd

This function calculates a 16 bit CRC over an additional memory range. The variable `start` is the CRC start value.

```c
__u16 sc_miscCrcContd ( __u8 *data, unsigned int len, __u16 start );
```

**Parameter**

- `data`  
  Pointer to the memory range.

- `len`  
  Number of bytes.

- `start`  
  CRC start value.

**Return Value**

16 bit CRC value.

**Please Note:** The system call `sc_miscCrcContd` is not supported in the SCIOPTA Compact Kernel.
3 System Call Reference

3.5 sc_m miscErrnoGet

This system call is used to get the process error number (errno) variable.

Each SCIOPTA process has an errno variable. This variable is used mainly by library functions to set the errno variable.

The errno variable will be copied into the observe messages if the process dies.

Please consult the SCIOPTA - Kernel, User’s Guide for more information about error handling and error hooks.

sc_errcode_t sc_m miscErrnoGet (void);

Parameter

none

Return Value

Read error code.
3.6 sc_miscErrnoSet

This system call is used to set the process error number (errno) variable.

Each SCIOPTA process has an errno variable. This variable is used mainly by library functions to set the errno variable.

The errno variable will be copied into the observe messages if the process dies.

Please consult the SCIOPTA - Kernel, User’s Guide for more information about error handling and error hooks.

```c
void sc_miscErrnoSet (  
   sc_errcode_t  err  
);
```

Parameter

err

User defined error code.

Return Value

none
3 System Call Reference

3.7 sc_miscError

This system call is used to call the error hooks with an user error.

The SCIOPTA error hooks is usually called when the kernel detects a system error. But the user can also call the error hook and including own error codes and additional information.

This system call will not return if there is no error hook. If an error hook is available the code of the error hook can decide to return or not.

Please consult the SCIOPTA - Kernel, User’s Guide for more information about error handling and error hooks.

```c
void sc_miscError ( sc_errcode_t err, sc_extra_t misc);
```

**Parameter**

**err**
User defined error code.

Bits 0, 1 and 2 of `err` defines if the error is fatal or not and if it is generated by the system, a module or a process.

**Bit 0:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fatal error on system level (SC_ERR_TARGET_FATAL)</td>
</tr>
<tr>
<td>0</td>
<td>No fatal error on system level</td>
</tr>
</tbody>
</table>

**Bit 1:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fatal error on module level (SC_ERR_MODULE_FATAL)</td>
</tr>
<tr>
<td>0</td>
<td>No fatal error on module level</td>
</tr>
</tbody>
</table>

**Bit 2:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fatal error on process level (SC_ERR_PROCESS_FATAL)</td>
</tr>
<tr>
<td>0</td>
<td>No fatal error on system level</td>
</tr>
</tbody>
</table>

Please consult also chapter 4.5 “Error Types” on page 4-5.

**misc**
Additional data to pass to the error hook.

**Return Value**

**none**
3.8  **sc_miscErrorHookRegister**

This system call will register an error hook.

Each time a system error occurs the error hook will be called if there is one installed.

Please consult the SCIOPTA - Kernel, User’s Guide for more information about error handling and error hooks.

```c
sc_errHook_t sc_miscErrorHookRegister (sc_errHook_t newhook);
```

**Parameter**

newhook

Function pointer to the hook. A 0 will remove and unregister the hook.

**Return Value**

Function pointer to the previous error hook. If no error hook was registered before, 0 will be returned.
3 System Call Reference

3.9 sc_moduleCreate

This system call is used to request the kernel daemon to create a module. The standard kernel daemon (sc_kerneld) needs to be defined and started at system configuration. Please consult the SCIOPTA - Kernel, User’s Guide for more information about kernel daemon.

SCIOPTA processes can be grouped into modules to improve system structure.

When creating a module the maximum number of pools and processes must be defined. There is a maximum number of 128 modules per SCIOPTA system possible.

Each module has also a priority which can range between 0 (highest) to 31 (lowest) priority. For process scheduling SCIOPTA uses a combination of the module priority and process priority called effective priority. The kernel determines the effective priority as follows:

\[
\text{Effective Priority} = \text{Module Priority} + \text{Process Priority}
\]

This technique assures that process with highest process priority (0) cannot disturb processes in modules with lower module priority (module protection).

Each module contains an init process with process priority=0 which will be created automatically.

If the module priority of the created module is higher than the effective priority of the caller the init process of the created module will be swapped in.

The start address of dynamically created modules must reside in RAM.

Please consult the SCIOPTA - Kernel, User’s Guide for more information about SCIOPTA modules.

```c
sc_moduleid_t sc_moduleCreate ( const char *name, void (*init) (void), sc_bufsize_t stacksize, sc_prio_t moduleprio, char *start, sc_modulesize_t size, sc_modulesize_t textsize, unsigned int max_pools, unsigned int max_procs );
```

Please Note: The system call sc_moduleCreate is not supported in the SCIOPTA Compact Kernel.
Parameter

name
Pointer to the name of the module to create.
The name is represented by an ASCII character string terminated by 0. The string can have up to 31 characters. Valid characters are A-Z, a-z, 0-9 and underscore. Specific international character and other control characters are not allowed.

init
Function pointer to the init process function. This is the address where the init process of the module will start execution.

stacksize
Stacksize of the init process in bytes.

moduleprio
Priority of the module.

start
Start address of the module in RAM.

size
Size of the module in bytes (RAM).
The minimum module size can be calculated according to the following formula (bytes):
size_mod = p * 128 + stack + pools + mcb + textsize
where:
p Number of static processes
stack Sum of stack sizes of all static processes
pools Sum of sizes of all message pools
mcb module control block (see below)
textsize code (see parameter textsize) below
The size of the module control block (mcb) can be calculated according to the following formula (bytes):
size_mcb = 96 + friends + hooks * 4 + P
where:
friend if friends are not used: 0
if friends are used 16 bytes
hooks number of hooks configured
P processor dependent, for PowerPC = 8, for all others = 0.
Please consult the configuration chapter of the target manual for information about friend and hook settings.
3 System Call Reference

**textsize**
Size of the module code in bytes (RAM). For non load modules this entry must be 0.

**max_pools**
Maximum number of pools in the module. The kernel will not allow to create more pools inside the module than stated here. Maximum value is 128.

**max_procs**
Maximum number of processes in the module. The kernel will not allow to create more pools inside the module than stated here. Maximum value is 16383.

**Return Value**
The module ID.
This system call is used to add a module to the friends of the caller. The caller accept the module in parameter `mid` as friend. The module is entered in the friend set of the caller.

In SCIOPTA each module can select friend modules. This has mainly consequences on whether messages will be copied or not.

Please consult the SCIOPTA - Kernel, User’s Guide for more information about the SCIOPTA module friend concept.

```c
void sc_moduleFriendAdd ( sc_moduleid_t mid );
```

**Parameter**

- **mid**
  The module ID of the new friend to add.

**Return Value**

- **none**
3 System Call Reference

3.11 sc_moduleFriendAll

**Please Note:** The system call `sc_moduleFriendAll` is not supported in the SCIOPTA Compact Kernel.

This system call is used to define all existing modules in a system as friend.

In SCIOPTA each module can select friend modules. This has mainly consequences on whether messages will be copied or not.

Please consult the SCIOPTA - Kernel, User’s Guide for more information about the SCIOPTA module friend concept.

```c
void sc_moduleFriendAll (void);
```

**Parameter**

*none*

**Return Value**

*none*
This system call is used to inform the caller if a module is a friend. The caller will be informed if the module in parameter `mid` is a friend. It returns if the module is member of the friend set of the caller.

In SCIOPTA each module can select friend modules. This has mainly consequences on whether messages will be copied or not.

Please consult the SCIOPTA - Kernel, User’s Guide chapter for more information about the SCIOPTA module friend concept.

```c
int sc_moduleFriendGet (sc_moduleid_t mid);
```

**Parameter**

`mid`

The ID of the module which will be checked if it is a friend or not.

**Return Value**

If the module is not a friend (not included in the friend set) the return value zero.

If the module is a friend (included in the friend set) the return value is one.
3.13 sc_moduleFriendNone

This system call is used to remove all modules as friends of the caller. All modules are removed in the friend set of the caller.

In SCIOPTA each module can select friend modules. This has mainly consequences on whether messages will be copied or not.

Please consult the SCIOPTA - Kernel, User's Guide for more information about the SCIOPTA module friend concept.

`void sc_moduleFriendNone (void);`

Parameter

none

Return Value

none
3.14 sc_moduleFriendRm

This system call is used to remove a module of the friends of the caller. The caller removes the module in parameter \texttt{mid} as friend. The module is removed in the friend set of the caller.

In SCIOPTA each module can select friend modules. This has mainly consequences on whether messages will be copied or not.

Please consult the SCIOPTA - Kernel, User’s Guide for more information about the SCIOPTA module friend concept.

\begin{verbatim}
void sc_moduleFriendRm (
    sc_moduleid_t mid
);
\end{verbatim}

**Parameter**

\texttt{mid}

The module ID of the old friend to remove.

**Return Value**

\texttt{none}
3.15 sc_moduleIdGet

**Please Note:** The system call `sc_moduleIdGet` is not supported in the SCIOPTA Compact Kernel.

This system call is used to get the ID of a module.

In contrast to the call `sc_procIdGet`, you can just give the name as parameter and not a path.

```c
sc_moduleId_t sc_moduleIdGet (const char *name);
```

**Parameter**

- **name**
  
  Pointer the 0 terminated name string of the module.

**Return Value**

If the module name was found the module ID is returned.

If the module name was not found the value `SC_NOSUCH_MODULE` is returned.
3  System Call Reference

3.16  sc_moduleInfo

Please Note: The system call sc_moduleInfo is not supported in the SCIOPTA Compact Kernel.

This system call is used to get a snap-shot of a module control block (mcb).

SCIOPTA maintains a module control block (mcb) per module and a process control block (pcb) per process which contains information about the module and process. System level debugger or run-time debug code can use this system call to get a copy of the control blocks.

The caller supplies a module control block structure in a local variable. The kernel will fill the structure with the control block data.

The structure content will reflect the module control block data at a certain moment which will be determined by the kernel. It is therefore a data snap-shot of which the exact time cannot be retrieved. You cannot directly access the module control blocks.

The structure of the module control block is defined in the module.h include file.

```c
int sc_moduleInfo (    
   sc_moduleid_t    mid,    
   sc_moduleInfo_t *info     
);
```

**Parameter**

mid
Module ID or process ID of which the control block data will be returned.

info
Pointer to a local structure of a module or process control block. This structure will be filled with the module control block data.

**Return Value**

If the module was found the return value is one and the info structure is filled with valid data.

If the module was not found the return value is zero.
3 System Call Reference

3.17 sc_moduleKill

Please Note: The system call sc_moduleKill is not supported in the SCIOPTA Compact Kernel.

This system call is used to dynamically kill a whole module.

The standard kernel daemon (sc_kerneld) needs to be defined and started at system configuration. Please consult the SCIOPTA - Kernel, User’s Guide for more information about kernel daemon.

All processes and pools in the module will be killed and removed.

The system call will return when the whole kill process is done.

```c
void sc_moduleKill (
    sc_moduleid_t         mid,
    flags_t              flags
);```

Parameter

mid
Module ID of the module to be killed and removed.

flags
This parameter can be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A cleaning up will be executed.</td>
</tr>
<tr>
<td>SC_MODULEKILL_KILL</td>
<td>No cleaning up will be requested.</td>
</tr>
</tbody>
</table>

Return Value

none
3.18  sc_moduleNameGet

Please Note: The system call sc_moduleNameGet is not supported in the SCIOPTA Compact Kernel.

This system call is used to get the name of a module. The name will be returned as a 0 terminated string.

```c
const char *sc_moduleNameGet ( sc_moduleid_t mid );
```

**Parameter**

mid

Module ID of which the name is requested.

**Return Value**

If the module was found the name string is returned.

If the module was not found the return value is NULL.
3 System Call Reference

3.19 sc_msgAcquire

This system call is used to change the owner of a message. The caller becomes the owner of the message.

The kernel will copy the message into a new message buffer allocated from the default pool on the following condition:

The message resides not in a pool of the callers module and the callers module is not friend to the module where the message resides.

In this case the message pointer (msgptr) will be modified.

Please use sc_msgAcquire with care. Transferring message buffers without proper ownership control by using sc_msgAcquire instead of transmitting and receiving messages with sc_msgTx and sc_msgRx will cause problems if you are killing processes.

void sc_msgAcquire ( 
    sc_msgptr_t msgptr 
); 

Parameter

msgptr

Pointer to the message buffer.

Return Value

none
3 System Call Reference

3.20 sc_msgAddrGet

This system call is used to get the process ID of the addressee of a message.

The kernel will examine the message buffer to determine the process to which the message was originally transmitted.

This system call is mainly used in communication software of distributed multi CPU systems (using connector processes). It allows to store the original addressee when you are forwarding a message by using the sc_msgTxAlias system call.

```c
sc_pid_t sc_msgAddrGet (sc_msgptr_t msgptr);
```

**Parameter**

- `msgptr`
  - Pointer to the message.

**Return Value**

The process ID of the addressee of the message is returned.
3.21 sc_msgAlloc

This system call will allocate a memory buffer of selectable size from a message pool.

SCIOPTA supports ownership of messages. The new allocated buffer is owned by the caller process. The owner of the message will change to the receiver process if the message is sent to another process. If you need to define a new owner without sending the message you could use the sc_msgAcquire system call. The call sc_msgAcquire must be used very carefully as this will pass messages around in a disorderly manner.

SCIOPTA is not returning the exact amount of bytes requested but will select one of a list of fixed buffer sizes which is large enough to contain the requested number of bytes. This list can contain 4, 8 or 16 fixed sizes which will be defined when a message pool is created. The content of the allocated message buffer is not initialized and can have any random value.

As SCIOPTA supports multiple pools the caller has to state the pool ID (plid) from where to allocate the message. The pool can only be in the same module as the caller process.

The caller can define how the system will respond to some limiting system states such as memory shortage in message pools and reply delays due to dynamic system behaviour (tmo).

```
sc_msg_t sc_msgAlloc (  
   sc_bufsize_t size,  
   sc_msgid_t id,  
   sc_poolid_t plid,  
   sc_ticks_t tmo
);
```

Parameter

**size**

The requested size of the message buffer.

**id**

Message ID which will be placed at the beginning of the data buffer of the message.

**plid**

Pool ID from where the message will be allocated.

This parameter can be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;pool id&gt;</td>
<td>Pool ID from where the message will be allocated.</td>
</tr>
<tr>
<td>SC_DEFAULT_POOL</td>
<td>Message will be allocated from the default pool. The default pool can be set by the system call sc_poolDefault.</td>
</tr>
</tbody>
</table>
### tmo

Allocation timing parameter:

This parameter can be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC_ENDLESS_TMO</td>
<td>Time-out is not used. Blocks and waits endless until a buffer is available from the message pool.</td>
</tr>
<tr>
<td>SC_NO_TMO</td>
<td>A NIL pointer will be returned if there is memory shortage in the message pool.</td>
</tr>
<tr>
<td>SC_FATAL_IF_TMO</td>
<td>A (fatal) kernel error will be generated if a message buffer of the requested size is not available.</td>
</tr>
<tr>
<td>0 &lt; tmo &lt; SC_TMO_MAX</td>
<td>Time-out value in system ticks. Alloc with time-out. Blocks and waits the specified number of ticks to get a message buffer.</td>
</tr>
</tbody>
</table>

### Return Value

The return value depends on the used tmo parameter when `sc_msgAlloc` was called.

- If the tmo parameter was SC_ENDLESS_TMO the pointer to the allocated buffer is returned.
- If the tmo parameter was SC_NO_TMO and if a buffer of the requested size is available the pointer to the allocated buffer is returned.
- If the tmo parameter was SC_NO_TMO and if a buffer of the requested size is not available a NIL pointer is returned.
- If the tmo parameter was a positive value (>0) and the system responds within the time-out period the pointer to the allocated buffer is returned.
- If the tmo parameter was a positive value (>0) and the system responds not within the time-out period (time-out expired) a NIL pointer is returned.
3.22 \texttt{sc\_msgAllocClr}

This system call works exactly the same as \texttt{sc\_msgAlloc} but it will initialize the data area of the message to 0. Please consult chapter 3.21 “\texttt{sc\_msgAlloc}” on page 3-23.

\begin{verbatim}
sc_msg_t sc_msgAllocClr (  
    sc_bufsize_t size,  
    sc_msgid_t id,  
    sc_poolid_t plidx,  
    sc_ticks_t tmo  
);  
\end{verbatim}

\textbf{Parameter}

Same as in chapter 3.21 “\texttt{sc\_msgAlloc}” on page 3-23.

\textbf{Return Value}

Same as in chapter 3.21 “\texttt{sc\_msgAlloc}” on page 3-23.
3.23  

**sc_msgFree**

This system call is used to return a message to the message pool if the message is no longer needed. Message buffers which have been returned can be used by other processes.

Only the owner of a message is allowed to free it by calling **sc_msgFree**. It is a fatal error to free a message owned by another process. If you have, for example transmitted a message to another process it is the responsibility of the receiving process to free the message.

Another process actually waiting to allocate a message of a full pool will become ready and therefore the caller process pre-empted on condition that:

1. the returned message buffer of the caller process has the same fixed size as the one of the waiting process and 
2. the priority of the waiting process is higher than the priority of the caller and 
3. the waiting process waits on the same pool as the caller will return the message.

```c
void sc_msgFree (
    sc_msgptr_t   msgptr
);
```

**Parameter**

msgptr

Pointer to the message.

**Return Value**

none
3 System Call Reference

3.24 sc_msgHookRegister

This system call will register a global or module message hook.

There can be one module message hook per module. If sc_msgHookRegister is called from within a module a module message hook will be registered. A global message hook will be registered when sc_msgHookRegister is called from the start hook function which is called before SCIOPTA is initialized.

Each time a message is sent or received (depending on the setting of parameter type) the module message hook of the caller will be called if such a hook exists. First the module and then the global message hook will be called.

\[
\text{sc_msgHook_t sc_msgHookRegister (}
\text{int type,}
\text{sc_msgHook_t newhook)}
\]

Parameter

type

This parameter can be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC_SET_MSGTX_HOOK</td>
<td>Registers a message transmit hook. Every time a message is sent, this hook will be called.</td>
</tr>
<tr>
<td>SC_SET_MSGRX_HOOK</td>
<td>Registers a message receive hook. Every time a message is received, this hook will be called.</td>
</tr>
</tbody>
</table>

newhook

Function pointer to the hook. A 0 will remove and unregister the hook.

Return Value

If a message hook was registered before this call the function pointer to the previous message hook will be returned.

If no message hook was registered before this call the return value is zero.
3.25  sc_msgOwnerGet

This system call is used to get the process ID of the owner of a message.

The kernel will examine the message buffer to determine the process who owns the message buffer.

```c
sc_pid_t sc_msgOwnerGet (sc_msgptr_t msgptr);
```

Parameter

msgptr

Pointer to the message.

Return Value

This system call returns the process ID of the owner of the message.
3.26   \texttt{sc\_msgPoolIdGet}

This system call is used to get the pool ID of a message.

When you are allocating a message with \texttt{sc\_msgAlloc} you need to give the ID of a pool from where the message will be allocated. During run-time you sometimes need to this information from received messages.

\begin{verbatim}
sc_poolid_t sc_msgPoolIdGet (sc_msgptr_t msgptr);
\end{verbatim}

\textbf{Parameter}

\begin{itemize}
  \item \texttt{msgptr}  \\
      Pointer to the message.
\end{itemize}

\textbf{Return Value}

If the message is in the same module than the caller, the pool ID where the message resides is returned.

If the message is \textbf{not} in the same module than the caller, the system call returns the value SC\_DEFAULT\_POOL.
System Call Reference

3.27 sc_msgRx

This system call is used to receive messages. The message queue of the caller will be searched for the desired messages.

If a message matching the conditions is received the kernel will return to the caller. If the message queue is empty or no wanted messages are available in the queue the process will be swapped out and another ready process with the highest priority will run. If a desired message arrives the process will be swapped in and the wanted list will be scanned again.

A pointer to an array (wanted) containing the messages (and/or process IDs) which will be scanned by sc_msgRx. The array must be terminated by 0.

A parameter flag (flag) controls different receiving methods:
1. The messages to be received are listed in a message ID array.
2. The array can also contain process IDs. In this case all messages sent by the listed processes are received.
3. You can also build an array of message ID and process ID pairs to receive specific messages sent from specific processes.
4. Also a message array with reversed logic can be given. In this case any message is received except the messages specified in the array.

If the pointer wanted to the array is NULL or the array is empty (contains only a zero element) all messages will be received.

The caller can also specify a time-out value tmo. The caller will not wait (swapped out) longer than the specified time. If the time-out expires the process will be made ready again and sc_msgRx will return with NULL.

\[
\text{sc_msg_t sc_msgRx (}
\text{sc_ticks_t tmo,}
\text{void *wanted,}
\text{int flag)}
\]

Parameter

tmo

Time-out parameter

This parameter can be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC_ENDLESS_TMO</td>
<td>Blocks and waits endless until the message is received.</td>
</tr>
<tr>
<td>SC_NO_TMO</td>
<td>No time-out, returns immediately. Must be set for interrupt and timer processes.</td>
</tr>
<tr>
<td>0 &lt; tmo &lt; SC_TMO_MAX</td>
<td>Time-out value in system ticks. Alloc with time-out. Blocks and waits the specified number of ticks to get a message buffer.</td>
</tr>
</tbody>
</table>
wanted

Pointer to the message (or pid) array.
This parameter can be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;ptr&gt;</td>
<td>Pointer to the message (or process ID) array.</td>
</tr>
<tr>
<td>SC_MSGRX_ALL</td>
<td>All messages will be received.</td>
</tr>
</tbody>
</table>

flag

More than one value can be defined and must be separated by OR instructions.
This member can be one or more of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC_MSGRX_MSGID</td>
<td>An array of wanted message IDs is given.</td>
</tr>
<tr>
<td>SC_MSGRX_PID</td>
<td>An array of process ID’s from where sent messages are received is given.</td>
</tr>
<tr>
<td>SC_MSGRX_NOT</td>
<td>An array of messages is given which will be excluded from receive.</td>
</tr>
<tr>
<td>SC_MSGRX_BOTH</td>
<td>An array of pairs of message ID’s and process ID’s are given to receive specific messages from specific transmitting processes.</td>
</tr>
</tbody>
</table>

Return Value

Pointer to the received message. The caller becomes owner of the received message.
If the time-out expires the process will be made ready again and the value NULL will be returned.
3 System Call Reference

3.28 sc_msgSndGet

This system call is used to get the process ID of the sender of a message.
The kernel will examine the message buffer to determine the process who has transmitted the message buffer.

```
sc_pid_t sc_msgSndGet (
   sc_msgptr_t    msgptr
);
```

Parameter

msgptr

   Pointer to the message.

Return Value

This call returns the process ID of the sender of the message. If the message was never sent, the process ID of the message owner will be returned.
3.29 \texttt{sc\_msgSizeGet}

This system call is used to get the requested size of a message. The requested size is the size of the message buffer when it was allocated. The actual kernel internal used fixed size might be larger.

\begin{verbatim}
sc_bufsize_t sc_msgSizeGet (sc_msgptr_t msgptr);
\end{verbatim}

\textbf{Parameter}

\texttt{msgptr}

Pointer to the message.

\textbf{Return Value}

The return value is the requested size of the message.
3 System Call Reference

3.30 sc_msgSizeSet

This system call is used to decrease the requested size of a message buffer.

The originally requested message buffer size is smaller (or equal) than the SCIOPTA internal used fixed buffer size. If the need of message data decreases with time it is sometimes favourable to decrease the requested message buffer size as well. Some internal operation are working on the requested buffer size.

The fixed buffer size for the message will not be modified.

```c
sc_bufsize_t sc_msgSizeSet (    
    sc_msgptr_t    msgptr,    
    sc_bufsize_t   newsz
);
```

Parameter

**msgptr**

Pointer to the message.

**newsz**

New requested size of the message buffer.

Return Value

The system call returns the new requested buffer size.

The old requested buffer size will be returned if there was a wrong request such as requesting a higher buffer size as the old one.

The system does not support increasing the buffer size.
3 System Call Reference

3.31 sc_msgTx

This system call is used to transmit a SCIOPTA message to a process (the addressee process).

Each SCIOPTA process has one message queue for messages which have been sent to the process. The sc_msgTx system call will enter the message at the end of the receivers message queue.

The caller cannot access the message buffer any longer as it is not any more the owner. The receiving process will become the owner of the message. NULL is loaded into the caller’s message pointer msgptr to avoid unintentional message access by the caller after transmitting.

The receiving process will be swapped in if it has a higher priority than the sending process.

If the addressee of the message resides not in the callers module and this module is not registered as a friend module the message will be copied before the transmit call will be executed. Messages which are transmitted across modules boundaries are always copied except if the modules are “friends”. To copy such a message the kernel will allocate a buffer from the pool of the module where the receiving process resides big enough to fit the message and copy the whole message. Message buffer copying depends on the friendship settings of the module where the buffer was originally allocated.

If the receiving process is not within the same target (CPU) as the caller the message will be sent to the connector process where the (distributed) receiving process is registered.

```c
void sc_msgTx (  
    sc_msgptr_t msgptr,  
    sc_pid_t addr,  
    flags_t flags );
```

Parameter

msgptr

Pointer to the message.

addr

The process ID of the addressee.

flags

Must be set to 0 (reserved for later use).

Return Value

none
3.32 sc_msgTxAlias

This system call is used to transmit a SCIOPTA message to a process by setting a process ID as sender.

The usual `sc_msgTx` system call sets always the calling process as sender. If you need to set another process ID as sender you can use this `sc_msgTxAlias` call.

This call is mainly used in communication software such as SCIOPTA connector processes where processes on other CPU’s are addressed. CONNECTOR processes will use this system call to enter the original sender of the other CPU.

Otherwise `sc_msgTxAlias` works the same way as `sc_msgTx`.

```c
void sc_msgTxAlias (  
    sc_msgptr_t msgptr,  
    sc_pid_t addr,  
    flags_t flags,  
    sc_pid_t alias  
);
```

Parameter

`msgptr`

Pointer to the message.

`addr`

The process ID of the addressee.

`flags`

Must be set to 0 (reserved for later use).

`alias`

The process ID specified as sender.

Return Value

none
3 System Call Reference

3.33 sc_poolCreate

This system call is used to create a new message pool inside the callers module.

Please consult the SCIOPTA - Kernel, User’s Guide for more information about message sizes and pools.

```c
sc_poolid_t sc_poolCreate(
    char *start,
    sc_plsize_t size,
    unsigned int nbufs,
    sc_bufsize_t *bufsize,
    const char *name
);
```

Parameter

start
Start address of the pool. If a 0 is given the kernel will automatically take the next free address in the module.

size
Size of the message pool. The minimum size must be the size of the maximum number of messages which ever are allocated at the same time plus the pool control block (pool_cb).

The size of the pool control block (pool_cb) can be calculated according to the following formula:

**Standard (32-Bit) kernel:**

```c
pool_cb = 68 + n * 20 + stat * n * 20
```

where:

- \( n \) = buffer sizes (4, 8 or 16)
- \( stat \) = process statistics or message statistics are used (1) or not used (0).

**Compact (16-Bit) kernel:**

```c
pool_cb = 15 + n * 6 + stat * n * 10 + P
```

where:

- \( n \) = buffer sizes (4 or 8)
- \( stat \) = process statistics or message statistics are used (1) or not used (0).
- \( P \) = processor dependent. For MSP430 = 1, for all others = 0.

Please consult the configuration chapter of the target manual for more information about statistics.

nbufs
The number of fixed buffer sizes. This can be 4, 8 or 16. It must always be lower or equal of the fixed buffer sizes which is defined for the whole system.
**bufsize**

Pointer to an array of the fixed buffer sizes in ascending order.

**name**

Pointer to the name of the pool to create.

The name is represented by a ASCII character string terminated by 0. The string can have up to 31 characters. Valid characters are A-Z, a-z, 0-9 and underscore. Specific international character and other control characters are not allowed.

**Return Value**

This system call returns the pool ID of the created message pool.
3.34  **sc_poolDefault**

This system call sets a message pool as default pool.

The default pool will be used by the `sc_msgAlloc` system call if the parameter for the pool to allocate the message from is defined as `SC_DEFAULT_POOL`.

Each process can set its default message pool by `sc_poolDefault`. The defined default message pool is stored inside the process control block. The initial default message pool at process creation is 0.

The default pool is also used if a message sent from another module needs to be copied.

```c
sc_poolid_t sc_poolDefault ( int idx );
```

**Parameter**

*idx*

This parameter can be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero or positive value</td>
<td>pool ID</td>
</tr>
<tr>
<td>-1</td>
<td>Request to return the ID of the default pool.</td>
</tr>
</tbody>
</table>

**Return Value**

This system call returns the old pool ID of the default message pool.
3 System Call Reference

3.35 sc_poolHookRegister

This system call will register a pool create or pool kill hook.

There can be one pool create and one pool kill hook per module.

Each time a pool is created or killed (depending on the setting of parameter type) the pool hook of the caller will be called if such a hook exists.

Syntax

```c
sc_poolHook_t sc_poolHookRegister (  
    int type,  
    sc_poolHook_t newhook);
```

Parameter

type

This parameter can be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC_SET_POOLCREATE_HOOK</td>
<td>Registers a pool create hook. Every time a pool is created, this hook will be called.</td>
</tr>
<tr>
<td>SC_SET_POOLKILL_HOOK</td>
<td>Registers a pool kill hook. Every time a pool is killed, this hook will be called.</td>
</tr>
</tbody>
</table>

typhoonhook

Function pointer to the hook. A 0 will remove and unregister the hook.

Return Value

This system call returns the function pointer to the previous pool create hook.

If no pool create hook was registered before the value zero will be returned.
3 System Call Reference

3.36 sc_poolIdGet

This system call is used to get the ID of a message pool.

In contrast to the call sc_procIdGet, you can just give the name as parameter and not a path.

```c
sc_poolid_t sc_poolIdGet (const char *name);
```

**Parameter**

**name**

Pointer the 0 terminated name string of the pool.

**Return Value**

If the pool name was found the system calls returns the pool ID.

If the pool name was not found the system calls returns the value SC_ILLEGAL_POOLID.
3 System Call Reference

3.37 sc_poolInfo

This system call is used to get a snap-shot of a pool control block.

SCIOPTA maintains a pool control block per pool which contains information about the pool. System level debugger or run-time debug code can use this system call to get a copy of the control block.

The caller supplies a pool control block structure in a local variable. The kernel will fill the structure with the control block data.

The structure content will reflect the pool control block data at a certain moment which will be determined by the kernel. It is therefore a data snap-shot of which the exact time cannot be retrieved. You cannot directly access the pool control blocks.

The structure of the pool control block is defined in the pool.h include file.

```c
int sc_poolInfo ( sc_moduleid_t mid, sc_poolid_t plid, sc_pool_cb_t *info );
```

**Parameter**

mid
Module ID where the pool resides of which the control block will be returned.

plid
ID of the pool of which the pool control block data will be returned.

info
Pointer to a local structure of a pool control block. This structure will be filled with the pool control block data.

**Return Value**

If the pool control block data was successfully retrieved the return value is nonzero.

If the system call fails and the pool control block data could not be retrieved the return value is zero.
3.38 sc_poolKill

This system call is used to kill a message pool.

A message pool can only be killed if all messages in the pool are freed (returned).

The killed pool memory can be reused later by a new pool if the size of the new pool is not exceeding the size of the killed pool.

Every process inside a module can kill a pool.

```c
void sc_poolKill (  
    sc_poolid_t plid 
);
```

Parameter

plid

Pool ID of the pool to be killed.

Return Value

none
3 System Call Reference

3.39  *sc_poolReset*

This system call is used to reset a message pool in its original state.

All messages in the pool must be freed and returned before a *sc_poolReset* call can be used.

The structure of the pool will be re-initialized. The message buffers in free-lists will be transformed back into unused memory. This “fresh” memory can now be used by *sc_msgAlloc* to allocate new messages.

Each process in a module can reset a pool.

```c
void sc_poolReset ( sc_poolid_t plid );
```

**Parameter**

*plid*

ID of the pool to reset.

**Return Value**

none
3 System Call Reference

3.40 sc_procCreate

Please Note: The system call sc_procCreate is only used in the SCIOPTA Compact Kernel.

This system call is used to create a process.

The maximum number of processes is defined at system configuration.

```
sc_pid_t sc_procCreate (sc_pdb_t *pdb, int state);
```

**Parameter**

pdb

Pointer to the process descriptor block variable of one of the following three types:

**For timer processes:**

```
typedef struct sc_pdbtim_s {
  sc_bufsize_t stacksize;
  sc_pcb_t *pcb;
  char *stack;
  void (*entry) (int);
  const char *name;
  int type;
  int initial_delay;
  int slice;
} sc_pdbtim_t;
```

**For interrupt processes:**

```
typedef struct sc_pdbint_s {
  sc_bufsize_t stacksize;
  sc_pcb_t *pcb;
  char *stack;
  void (*entry) (int);
  const char *name;
  int type;
  int vector;
} sc_pdbint_t;
```

**For prioritized processes:**

```
typedef struct sc_pdbprio_s {
  sc_bufsize_t stacksize;
  sc_pcb_t *pcb;
  char *stack;
  void (*entry) (void);
  const char *name;
  int type;
  __u8 prio;
} sc_pdbprio_t;
```
Parameters used by all three process types:

**stacksize**
Stacksize of the created process in bytes. Add 38 bytes for the process control block (pcb) and 32 bytes for the process name to your process stack to calculate the minimum stacksize.

Add another 4 bytes if you have enabled C_LINE debugging in the systems configuration.

For the HCS12 kernel and if page memory is used you need to add one additional byte.

For the M16C kernel you need to add another 2 bytes.

**pcb**
Pointer to the process control block. This must be reserved by the user. The kernel will use it for the pcb data.

**stack**
Pointer to the process stack. This stack area must be reserved by the user.

**entry**
Function pointer to the process function. This is the address where the created process will start execution.

**name**
Pointer to the process name.

The name is represented by a ASCII character string terminated by 0. The string can have up to 31 characters. Valid characters are A-Z, a-z, 0-9 and underscore. Specific international character and control characters are not allowed.

**type**
This parameter can be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC_PDB_TIM</td>
<td>for timer processes.</td>
</tr>
<tr>
<td>SC_PDB_INT</td>
<td>for interrupt processes.</td>
</tr>
<tr>
<td>SC_PDB_PRIO</td>
<td>for prioritized processes.</td>
</tr>
<tr>
<td>SC_PDB_USRINT</td>
<td>for user interrupt processes.</td>
</tr>
</tbody>
</table>

Additional parameters for timer processes:

**initial_delay**
Initial delay in ticks before the first time call to the timer process.

**slice**
Period of time between calls to the timer process in ticks.

Additional parameter for interrupt processes:

**vector**
Interrupt vector connected to the created interrupt process. This is CPU-dependent.

Additional parameter for prioritized processes:

**prio**
The priority of the process which can be from 0 to 31. 0 is the highest priority.
3 System Call Reference

state

Process state after creation. This parameter is not used by interrupt processes.
This parameter can be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC_PDB_STATE_RUN</td>
<td>The process will be in READY state. It is ready to run and will be swapped-in if it has the highest priority of all READY processes.</td>
</tr>
<tr>
<td>SC_PDB_STATE_STP</td>
<td>The process is stopped. Use the sc_procStart system call to start the process.</td>
</tr>
</tbody>
</table>

Return Value

This system call returns the process ID of the created process.
3 System Call Reference

3.41 sc_procDaemonRegister

Please Note: The system call `sc_procRegisterDaemon` is not supported in the SCIOPTA Compact Kernel.

This system call is used to register a process daemon.

The process daemon manages process names in a SCIOPTA system. If a process calls `sc_procIdGet` the kernel will send a `sc_procIdGet` message to the process daemon. The process daemon will search the process name list and return the corresponding process ID to the kernel if found.

There can only be one process daemon per SCIOPTA system.

The standard process daemon `sc_procd` is included in the SCIOPTA kernel. This process daemon needs to be defined and started at system configuration as a static process. Please consult the SCIOPTA - Kernel, User’s Guide for more information about process daemon.

```c
int sc_procRegisterDaemon (void);
```

Parameter

none

Return Value

If the process daemon was successfully installed the return value is zero.

If the system call fails and the process daemon could not be installed the return value is nonzero.
3.42 sc_procDaemonUnregister

Please Note: The system call sc_procUnregisterDaemon is not supported in the SCIOPTA Compact Kernel.

This call is used by a process daemon to unregister.
The name list of the daemon will be removed and messages still owned by the daemon will be freed.
A statically installed process daemon cannot be unregistered.

void sc_procUnregisterDaemon (void);

Parameter
none

Return Value
none
3 System Call Reference

3.43 sc_procHookRegister

This system call will register a process hook of the type defined in parameter **type**. The type can be a create hook, kill hook or swap hook.

Each time a process will be created the create hook will be called if there is one installed.

Each time a process will be killed the kill hook will be called if there is one installed.

Each time a process swap is initiated by the kernel the swap hook will be called if there is one installed.

```c
sc_procHook_t sc_procHookRegister ( int type,
                                    sc_procHook_t newhook);
```

**Parameter**

**type**

This parameter can be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC_SET_PROCCREATE_HOOK</td>
<td>Registers a process create hook. Every time a process is created, this hook will be called.</td>
</tr>
<tr>
<td>SC_SET_PROCKILL_HOOK</td>
<td>Registers a process kill hook. Every time a process is killed, this hook will be called.</td>
</tr>
<tr>
<td>SC_SET_PROCSWAP_HOOK</td>
<td>Registers a process swap hook. Every time a process swap is initiated by the kernel, this hook will be called.</td>
</tr>
</tbody>
</table>

**newhook**

Function pointer to the hook. A 0 will remove and unregister the hook.

**Return Value**

This system call returns the function pointer to the previous process hook.

If no process hook was registered before the value zero will be returned.
3 System Call Reference

3.44 sc_procIdGet

This call is used to get the process ID of a process by providing the name of the process.

In SCIOPTA processes are organized in systems (CPUs) and modules within systems. There is always at least one module called system module (module 0). Depending where the process resides (system, module) not only the process name needs to be supplied but also the including system and module name.

This call forwards the request to the process daemon. The standard process daemon (sc_procd) needs to be defined and started at system configuration. Please consult the SCIOPTA - Kernel, User’s Guide for more information about process daemon.

In the Compact Kernel the process daemon needs not to be defined and started. This is done inside the kernel. Compact Kernels do not allow a timeout (tmo).

sc_pid_t sc_procIdGet (const char *path, sc_ticks_t tmo);

Parameter

path

Pointer to the path including the name of the process.

If the process resides within the caller’s module:
path::=process_name

If the process resides in the system module of the caller’s target:
path::=/process_name

If the process resides in the system module of an external target:
path::=/<system_name>/process_name

If the process resides in another than the system module of the caller’s target:
path::=/<module_name>/process_name

If the process resides in another than the system module of an external target:
path::=/<system_name>/<module_name>/process_name

If path is NULL (and the parameter tmo is SC_NO_TMO) this system call returns the current process ID (the process ID of the caller).
tmo

Time to wait for a response in ticks. This parameter is not allowed if asynchronous timeout is disabled at system configuration (sconf). Please consult the configuration chapter of the kernel target manual for your processor.

This parameter can be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC_NO_TMO</td>
<td>No time-out, returns immediately.</td>
</tr>
<tr>
<td>0 &lt; tmo &lt; SC_TMO_MAX</td>
<td>Time-out value in system ticks.</td>
</tr>
</tbody>
</table>

Return Value

If the process was found within the tmo time period the system call returns the process ID of the found process.

If the process was not found within the tmo time period the system call returns the value SC_ILLEGAL_PID.
3  System Call Reference

3.45  sc_procIntCreate

Please Note: The system call sc_procIntCreate is not supported in the SCIOPTA Compact Kernel.

This system call is used to request the kernel daemon to create an interrupt process. The standard kernel daemon (sc_kerneld) needs to be defined and started at system configuration. Please consult the SCIOPTA - Kernel, User’s Guide for more information about kernel daemon.

The process will be created within the callers module.

The maximum number of processes for a specific module is defined at module creation.

```c
sc_pid_t sc_procIntCreate (  
    const char *name,  
    void (*entry) (int),  
    sc_bufsize_t stacksize,  
    int vector,  
    sc_prio_t prio,  
    int state,  
    sc_poolid_t plid  
);  
```

**Parameter**

**name**

Pointer to the name of the interrupt process to create.

The name is represented by a ASCII character string terminated by 0. The string can have up to 31 characters. Valid characters are A-Z, a-z, 0-9 and underscore. Specific international character and other control characters are not allowed.

**entry**

Function pointer to the process function. This is the address where the created process will start execution.

**stacksize**

Stacksize of the created process in bytes. The kernel will use one of the fixed message buffer sizes from the message pool which is big enough to include the stack. Add 128 bytes for the process control block (pcb) and 32 bytes for the process name to your process stack to calculate the minimum stacksize.

**vector**

Interrupt vector connected to the created interrupt process. This is CPU-dependent.
prio
   Must be set to 0 (reserved for later use).

state
   Must be set to 1 (reserved for later use).

plid
   Pool ID from where the message buffer (which holds the stack and the pcb) will be allocated.

Return Value

This system call returns the process ID of the created interrupt process.
3 System Call Reference

3.46 sc_procKill

This system call is used to request the kernel daemon to kill a process.

The standard kernel daemon (sc_kerneld) needs to be defined and started at system configuration. Please consult the SCIOPTA - Kernel, User’s Guide for more information about kernel daemon.

Please Note:
There is no kernel daemon to declare and to define in the SCIOPTA Compact Kernel. This function is defined and started automatically by the kernel.

Any process type (prioritized, interrupt, timer) can be killed. No external processes (on a remote CPU) can be killed.

If a cleaning-up is executed (depending on the flag parameter) all message buffers owned by the process will be returned to the message pool. If an observe is active on that process the observe messages will be sent to the observing processes. The sc_procKill system calls returns before the cleaning work begins. A significant time can elapse before a possible observe message is posted.

```c
void sc_procKill ( 
    sc_pid_t  pid, 
    flags_t   flag);
```

Parameter

pid
Process ID of the process to be killed.

flag
This parameter can be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A cleaning up will be executed.</td>
</tr>
<tr>
<td>SC_PROCKILL_KILL</td>
<td>No cleaning up will be requested.</td>
</tr>
</tbody>
</table>

Return Value

none
3 System Call Reference

3.47 sc_procNameGet

This call is used to get the full name of a process.

The name will be returned inside a SCIOPTA message buffer which is allocated by the kernel. The kernel sets the caller as owner of the message.

The user must free the message after use.

```c
sc_msg_t sc_procNameGet (sc_pid_t pid);
```

Parameter

pid

Process ID of the process where the name is requested.

Return Value

If the pointer to a message buffer is owned by the caller and the message is of type `sc_procNameGetMsgReply_t` and the message ID is `SC_PROCNAMEGETMSG_REPLY` the return value is nonzero. The message data contains the 0 terminated ASCII name string of the process. The message is defined in the `sciopta.msg` include file.

```c
typedef struct sc_procNameGetMsgReply_s {
  sc_msgid_t id;
  sc_errcode_t error;
  char target[SC_MODULE_NAME_SIZE+1];
  char module[SC_MODULE_NAME_SIZE+1];
  char process[SC_PROC_NAME_SIZE+1];
} sc_procNameGetMsgReply_t;
```

If the system contains an error hook and if the process (pid) does not exist (any more) the return value is zero.
3.48 sc_procObserve

Please Note: The system call sc_procObserve is not supported in the SCIOPTA Compact Kernel.

This system call is used to supervise a process.

The sc_procObserve system call will request the message to be sent back if the given process dies (process supervision). If the supervised process disappears from the system (process ID) the kernel will send the requested and registered message to the supervisor process.

The process to supervise can be external (in another CPU).

```c
void sc_procObserve (  
   sc_msgptr_t msgptr,  
   sc_pid_t pid  
);
```

Parameter

msgptr

Pointer to the message which will be returned if the supervised process disappears. The message must be of the following type:

```c
struct err_msg {  
   sc_msgid_t id;  
   sc_errcode_t error;  
   /* user defined data */
};
```

pid

Process ID of the process which will be supervised.

Return Value

none
3 System Call Reference

3.49 sc_procPathGet

This call is used to get the full path of a process.

The path will be returned inside a SCIOPTA message buffer which is allocated by the kernel. The kernel sets the caller as owner of the message.

The user must free the message after use.

```c
sc_msg_t sc_procPathGet (sc_pid_t pid,
                          flags_t flags);
```

**Parameter**

**pid**
Process ID of the process where the path is requested.

**flags**
This parameter can be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nonzero</td>
<td>The full path is returned: <code>//&lt;system_name&gt;/&lt;module_name&gt;/&lt;process_name</code></td>
</tr>
<tr>
<td>Zero</td>
<td>The short path is returned: <code>/&lt;module_name&gt;/&lt;process_name</code></td>
</tr>
</tbody>
</table>

**Return Value**

If the pointer to a message buffer is owned by the caller and the message is of type `sc_procPathGetMsgReply_t` and the message ID is SC_PROCPATHGETMSG_REPLY the return value is nonzero. The message data contains the 0 terminated ASCII name string of the process. The message is defined in the sciopta.msg include file.

```c
typedef struct sc_procPathGetMsgReply_s {
  scmsgid_t id;
  sc_pid_t pid;
  sc_errcode_t error;
  char path[1];
} sc_procPathGetMsgReply_t;
```

If the system contains an error hook and if the process (pid) does not exist (any more) the return value is zero.
3.50 sc_procPpidGet

Please Note: The system call `sc_procPpidGet` is not supported in the SCIOPTA Compact Kernel.

This call is used to get the process ID of the parent (creator) of a process.

```c
sc_pid_t sc_procPpidGet ( sc_pid_t pid );
```

**Parameter**

*pid*

This parameter can be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;pid&gt;</code></td>
<td>The parent process ID of any (given) process (pid) will be returned.</td>
</tr>
<tr>
<td>SC_CURRENT_PID</td>
<td>The parent process ID of the callers process will be returned</td>
</tr>
</tbody>
</table>

**Return Value**

If the parent process exists this call returns the process ID of the parent process.

If the parent process does no longer exist the value SC_ILLEGAL_PID will be returned.
3 System Call Reference

3.51 sc_procPrioCreate

Please Note: The system call sc_moduleNameGet is not supported in the SCIOPTA Compact Kernel.

This system call is used to request the kernel daemon to create a prioritized process. The standard kernel daemon (sc_kerneld) needs to be defined and started at system configuration. Please consult the SCIOPTA - Kernel, User’s Guide for more information about kernel daemon.

The process will be created within the callers module.

The maximum number of processes for a specific module is defined at module creation.

```c
sc_pid_t sc_procPrioCreate (  
    const char *name,  
    void (*entry) (void),  
    sc_bufsize_t stacksize,  
    sc_ticks_t slice,  
    sc_prio_t prio,  
    int state,  
    sc_poolid_t plid  
);
```

Parameter

name

Pointer to the name of the prioritized process to create.

The name is represented by a ASCII character string terminated by 0. The string can have up to 31 characters. Valid characters are A-Z, a-z, 0-9 and underscore. Specific international character and other control characters are not allowed.

entry

Function pointer to the process function. This is the address where the created process will start execution.

stacksize

Stacksize of the created process in bytes. The kernel will use one of the fixed message buffer sizes from the message pool which is big enough to include the stack. Add 128 bytes for the process control block (pcb) and 32 bytes for the process name to your process stack to calculate the minimum stacksize.

slice

Must be set to 0 (reserved for later use).

prio

The priority of the process which can be from 0 to 31. 0 is the highest priority.
state

Process state after creation.

This parameter can be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC_PDB_STATE_RUN</td>
<td>The process will be on READY state. It is ready to run and will be swapped-in if it has the highest priority of all READY processes.</td>
</tr>
<tr>
<td>SC_PDB_STATE_STP</td>
<td>The process is stopped. Use the sc_procStart system call to start the process.</td>
</tr>
</tbody>
</table>

plid

Pool ID from where the message buffer (which holds the stack and the pcb) will be allocated.

Return Value

This system call returns the process ID of the created prioritized process.
3.52  **sc_procPrioGet**

This process is used to get the priority of a prioritized process.

In SCIOPTA the priority ranges from 0 to 31. 0 is the highest and 31 the lowest priority.

```c
sc_prio_t sc_procPrioGet (sc_pid_t pid);
```

**Parameter**

**pid**

This parameter can be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;pid&gt;</td>
<td>The priority of any (given) process (pid) will be returned.</td>
</tr>
<tr>
<td>SC_CURRENT_PID</td>
<td>The priority of the callers process will be returned.</td>
</tr>
</tbody>
</table>

**Return Value**

This function call returns the priority of the prioritized process.
3 System Call Reference

3.53 sc_procPrioSet

This call is used to set the priority of a process.

Only the priority of the caller’s process can be set and modified.

If the new priority is lower to other ready processes the kernel will initiate a context switch and swap in the process with the highest priority.

Init processes are treated specifically. An init process is the first process in a module and does always exist. An init process can set its priority on level 32 (this is the only process which can have a priority of 32). This will redefine the process and it becomes an idle process. The init process should always set its priority to 32 after it has accomplished its initialization work. The idle process will be called by the kernel if there are no processes ready for getting the CPU (all are in a waiting state).

```c
void sc_procPrioSet (  
    sc_prio_t      prio  
);  
```

Parameter

prio

The new priority of the caller’s process (0 .. 31).

Return Value

none
3.54 sc_procSchedLock

This system call will lock the scheduler and return the number of times it has been locked before.

SCIOPTA maintains a scheduler lock counter. If the counter is 0 scheduling is activated. Each time a process calls `sc_procSchedLock` the counter will be incremented.

Interrupts are not blocked if the scheduler is blocked by `sc_procSchedLock`.

**Syntax**

```c
int sc_procSchedLock (void);
```

**Parameter**

none

**Return Value**

This system call returns the value of the internal scheduler lock counter. This corresponds to the number of times the scheduler has been locked.
3.55  \texttt{sc\_procSchedUnlock}

This system call will unlock the scheduler.

SCIOPTA maintains a scheduler lock counter. Each time a process calls \texttt{sc\_procSchedUnlock} the counter will be decremented. If the counter reaches a value of 0 the SCIOPTA scheduler is called and activated. The ready process with the highest priority will be swapped in.

It is illegal to unlock a not blocked scheduler.

\begin{verbatim}
void sc_procSchedUnlock (void);
\end{verbatim}

Parameter

none

Return Value

none
3.56 sc_procSliceGet

This call is used to get the time slice of a timer process.

The time slice is the period of time between calls to the timer process in ticks.

```c
sc_ticks_t sc_procSliceGet (
    sc_pid_t pid
);
```

**Parameter**

*pid*

This parameter can be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;pid&gt;</td>
<td>The time slice of any (given) process (pid) will be returned.</td>
</tr>
<tr>
<td>SC_CURRENT_PID</td>
<td>The time slice of the callers process will be returned.</td>
</tr>
</tbody>
</table>

**Return Value**

This system call returns the period of time between calls to the timer process in ticks.
3 System Call Reference

3.57 sc_procSliceSet

This call is used to set the time slice of a timer process. The modified time slice will become active after the current time slice expired or if the timer gets started. It can only be activated after the old time slice has elapsed.

```c
void sc_procSliceSet (    
    sc_pid_t pid,    
    sc_ticks_t slice
);
```

Parameter

**pid**

This parameter can be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;pid&gt;</td>
<td>Process ID of any (given) timer process.</td>
</tr>
<tr>
<td>SC_CURRENT_PID</td>
<td>Callers timer process ID.</td>
</tr>
</tbody>
</table>

**slice**

New period of time between calls to the timer process in ticks.

Return Value

**none**
### 3.58 sc_procStart

This system call will start a prioritized or timer process.

SCIOPTA maintains a start/stop counter per process. If the counter is >0 the process is stopped. Each time a process calls `sc_procStart` the counter will be decremented. If the counter has reached the value of 0 the process will start.

If the started process is a prioritized process and its priority is higher than the priority of the currently running process, it will be swapped in and the current process swapped out.

If the started process is a timer process, it will be entered into the timer list with its time slice.

It is illegal to start a process which was not stopped before.

```c
void sc_procStart (
   sc_pid_t   pid
);
```

#### Parameter

**pid**

Process ID of the process to start.

#### Return Value

**none**
3 System Call Reference

3.59 sc_procStop

This system call will stop a prioritized or timer process.

SCIOPTA maintains a start/stop counter per process. If the counter is >0 the process is stopped. Each time a process calls `sc_procStop` the counter will be incremented.

If the stopped process is the currently running prioritized process, it will be halted and the next ready process will be swapped in.

If a timer process will be stopped, it will immediately removed from the timer list and the system will not wait until the current time slice expires.

```c
void sc_procStop (  
    sc_pid_t    pid  
);
```

Parameter

pid

Process ID of the process to stop.

Return Value

none
3 System Call Reference

3.60 sc_procTimCreate

Please Note: The system call sc_procTimCreate is not supported in the SCIOPTA Compact Kernel.

This system call is used to request the kernel daemon to create a timer process. The standard kernel daemon (sc_kerneld) needs to be defined and started at system configuration. Please consult the SCIOPTA - Kernel, User’s Guide for more information about kernel daemon.

The process will be created within the callers module.

The maximum number of processes for a specific module is defined at module creation.

```
csc sexuality t  sc_procTimCreate ( 
  const char *name, 
  void (*entry) (int), 
  sc_bufsize_t stacksize, 
  sc_ticks_t period, 
  sc_ticks_t initdelay, 
  int state, 
  sc_poolid_t plid
);```

Parameter

name

Pointer to the name of the timer process to create.

The name is represented by a ASCII character string terminated by 0. The string can have up to 31 characters. Valid characters are A-Z, a-z, 0-9 and underscore. Specific international character and other control characters are not allowed.

entry

Function pointer to the process function. This is the address where the created process will start execution.

stacksize

Stacksize of the created process in bytes. The kernel will use one of the fixed message buffer sizes from the message pool which is big enough to include the stack. Add 128 bytes for the process control block (pcb) and 32 bytes for the process name to your process stack to calculate the minimum stacksize.

period

Period of time between calls to the timer process in ticks.

initdelay

Initial delay in ticks before the first time call to the timer process.
3 System Call Reference

state

Process state after creation.
This parameter can be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC_PDB_STATE_RUN</td>
<td>The process will be on READY state. It is ready to run and will be</td>
</tr>
<tr>
<td></td>
<td>swapped-in if it has the highest priority of all READY processes.</td>
</tr>
<tr>
<td>SC_PDB_STATE_STP</td>
<td>The process is stopped. Use the sc_procStart system call to start the</td>
</tr>
<tr>
<td></td>
<td>process.</td>
</tr>
</tbody>
</table>

plid

Pool ID from where the message buffer (which holds the stack and the pcb) will be allocated.

Return Value

This system call returns the process ID of the created timer process.
3.61 sc_procUnobserve

Please Note: The system call sc_procUnobserve is not supported in the SCIOPTA Compact Kernel.

This system call is used to cancel an installed supervision of a process.
The message given by the sc_procObserve system call will be freed by the kernel.

```c
void sc_procUnobserve (sc_pid_t pid, sc_pid_t observer);
```

**Parameter**

**pid**
Process ID of the process which is supervised.

**observer**
Process ID of the observer process.

**Return Value**

none
3.62  sc_procUsrIntCreate

Please Note: The system call sc_procUsrIntCreate is not supported in the SCIOPTA Compact Kernel.

This system call is used to request the kernel daemon to create a user interrupt process. The standard kernel daemon (sc_kerneld) needs to be defined and started at system configuration. Please consult the SCIOPTA - Kernel, User’s Guide for more information about kernel daemon.

The process will be created within the callers module.

The maximum number of processes for a specific module is defined at module creation.

```
sc_pid_t sc_procUsrIntCreate (
    const char *name,
    void (*entry) (int),
    sc bufsize_t stacksize,
    int vector,
    sc prio_t prio,
    int state,
    sc poolid_t plid
);
```

Parameter

**name**

Pointer to the name of the user interrupt process to create.

The name is represented by a ASCII character string terminated by 0. The string can have up to 31 characters. Valid characters are A-Z, a-z, 0-9 and underscore. Specific international character and other control characters are not allowed.

**entry**

Function pointer to the process function. This is the address where the created process will start execution.

**stacksize**

Stacksize of the created process in bytes. The kernel will use one of the fixed message buffer sizes from the message pool which is big enough to include the stack. In the same message buffer the kernel will include the process control block pcb.

**vector**

Interrupt vector connected to the created user interrupt process. This is CPU-dependent.
3 System Call Reference

prio
Must be set to 0 (reserved for later use).

state
Must be set to 1 (reserved for later use).

plid
Pool ID from where the message buffer (which holds the stack and the pcb) will be allocated.

Return Value
This system call returns the process ID of the created user interrupt process.
3.63 sc_procVarDel

This system call is used to remove a process variable from the process variable data area.
Please consult the SCIOPTA - Kernel, User’s Guide for more information about process variables.

```c
int sc_procVarDel (  
    sc_tag_t     tag
);
```

**Parameter**

**tag**

User defined tag of the process variable which was set by the `sc_procVarSet` call.

**Return Value**

If the process variable was successfully removed, the return value is nonzero.
If the system call fails and the process variable could not be removed the return value is zero.
3 System Call Reference

3.64 sc_procVarGet

This system call is used to read a process variable.

Please consult the SCIOPTA - Kernel, User’s Guide for more information about process variables.

```c
int sc_procVarGet(
    sc_tag_t tag,
    sc_var_t *value
);
```

Parameter

tag

User defined tag of the process variable which was set by the `sc_procVarSet` call.

value

Pointer to the variable where the process variable will be stored.

Return Value

If the process variable was successfully read, the return value is nonzero.

If the system call fails and the process variable could not be read the return value is zero.
3 System Call Reference

3.65 sc_procVarInit

This system call is used to setup and initialize a process variable area.

The user needs to allocate a message with the size of

\[ \text{sizeof (sc	extunderscore local	extunderscore t)} \times \text{size} \]

The pointer to the allocated message needs to be included as parameter in \texttt{sc\_procVarInit}.

Please consult the SCIOPTA - Kernel, User’s Guide for more information about process variables.

```c
void sc_procVarInit (  
    sc_msgptr_t varpool,  
    unsigned int size  
);
```

Parameter

\texttt{varpool}

Pointer to the message buffer holding the process variables.

\texttt{size}

Maximum number of process variables + 1.

Return Value

\texttt{none}
3.66 sc_procVarRm

This system call is used to remove a whole process variable area.

Please consult the SCIOPTA - Kernel, User’s Guide for more information about process variables.

```
sc_msg_t sc_procVarRm (void);
```

Parameter

none

Return Value

This system call returns the pointer to the message buffer holding the process variables.
3.67 sc_procVarSet

This system call is used to define or modify a process variable.

Please consult the SCIOPTA - Kernel, User’s Guide for more information about process variables.

```c
int sc_procVarSet (
    sc_tag_t tag,
    sc_var_t value
);
```

**Parameter**

- `tag` 
  User defined tag for the process variable.

- `value` 
  Value of the process variable.

**Return Value**

If the process variable was successfully defined or modified, the return value is nonzero.

If the system call fails and the process variable could not be defined or modified, the return value is zero.
3.68 sc_procVectorGet

This system call is used to get the interrupt vector of the caller.
The interrupt vector will only be returned if `sc_procVectorGet` is called from an interrupt process.
It is not an error to call `sc_procVectorGet` from other process types but the return value will be meaningless.

```c
int sc_procVectorGet (void);
```

Parameter

none

Return Value

This system call returns the interrupt vector if called within an interrupt process.
3.69  sc_procYield

This system call is used to yield the CPU to the next ready process within the current process's priority group.

`void sc_procYield (void);`

**Parameter**

none

**Return Value**

none
### 3.70 sc_sleep

This call is used to suspend the calling process for a defined time. The requested time must be given in number of system ticks.

The calling process will get into a waiting state and swapped out. After the time-out has elapsed the process will become ready again and will be swapped in if it has the highest priority of all ready processes.

The process will be waiting for at least the requested time minus one system tick.

```c
void sc_sleep (sc_ticks_t tmo);
```

#### Parameter

- **tmo**
  
  Number of system ticks to wait.

#### Return Value

none
3 System Call Reference

3.71 sc_tick

This function calls directly the kernel tick function and advances the kernel tick counter by 1.

The kernel maintains a counter to control the timing functions. The timer needs to be incremented in regular intervals.

If the processor contains an on-chip timer the kernel uses it by default. The on-chip timer is set-up by the kernel automatically at start-up and all parameters are defined in the SCIOPTA configuration utility. In this case the user does not need to call sc_tick.

If the processor does not have an on-chip timer or the user does not want to use it, sc_tick must be called explicitly by the user from within an user interrupt process. The user is responsible to write the user interrupt process and to setup the timer chip to define the requested tick interval.

void sc_tick (void);

Parameter

none

Return Value

none
3.72  sc_tickGet

This call is used to get the actual kernel tick counter value. The number of system ticks from the system start are returned.

```
sc_time_t sc_tickGet (void);
```

**Parameter**

none

**Return Value**

This system call returns the number of system ticks of the kernel tick counter.
3.73  

This system call is used to set the current system tick length in micro seconds.

```
__u32 sc_tickLength ( __u32 tl );
```

**Parameter**

*tl*

This parameter can be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero</td>
<td>The current tick length will just be returned without modifying it.</td>
</tr>
<tr>
<td>&lt;tick_length&gt;</td>
<td>The tick length in micro seconds</td>
</tr>
</tbody>
</table>

**Return Value**

This system call returns the tick length in microseconds.
3 System Call Reference

3.74 sc_tickMs2Tick

This system call is used to convert a time from milliseconds into system ticks.

```c
sc_time_t sc_tickMs2Tick ( __u32 ms );
```

**Parameter**

- **ms**
  - Time in milliseconds

**Return Value**

This system call returns the time in system ticks.
3.75  **sc_tickTick2Ms**

This system call is used to convert a time from system ticks into milliseconds.

```c
__u32 sc_tickTick2Ms ( sc_ticks_t t );
```

**Parameter**

- `t`  
  Time in system ticks.

**Return Value**

This system call returns the time in milliseconds.
3.76  **sc_tmoAdd**

This system call is used to request a time-out message from the kernel after a defined time.

The caller needs to allocate a message and include the pointer to this message in the call. The kernel will send this message back to the caller after the time has expired.

This is an asynchronous call, the caller will not be blocked.

The registered time-out can be cancelled by the **sc_tmoRm** call before the time-out has expired.

```c
sc_tmoid_t sc_tmoAdd (  
   sc_ticks_t   tmo, 
   sc_msgptr_t  msgptr  
);  
```

**Parameter**

- **tmo**
  
  Number of system tick after which the message will be sent back by the kernel.

- **msgptr**
  
  Pointer to the message which will be sent back by the kernel after the elapsed time.

**Return Value**

This system call returns the time-out ID which could be used later to cancel the time-out.
3.77 sc_tmoRm

This system call is used to remove a time-out before it is expired.

If the process has already received the time-out message and the user still tries to cancel the time-out with the sc_tmoRm call, the kernel will generate a fatal error.

```c
sc_msg_t sc_tmoRm (sc_tmoid_t *id);
```

Parameter

id

Time-out ID which was given when the time-out was registered by the sc_tmoAdd call.

Return Value

This system call returns the pointer to the time-out message which was defined at registering it by the sc_tmoAdd call.
3 System Call Reference

3.78 sc_trigger

This system call is used to activate a process trigger.

The trigger value of the addressed process trigger will be incremented by 1. If the trigger value becomes greater than zero the process waiting at the trigger will become ready and swapped in if it has the highest priority of all ready processes.

Please consult the SCIOPTA - Kernel, User’s Guide for more information about SCIOPTA trigger.

```c
void sc_trigger ( sc_pid_t pid );
```

**Parameter**

**pid**

ID of the process which trigger will be activated.

**Return Value**

This system call returns the incremented trigger value.
3.79  sc_triggerValueGet

This system call is used to get the value of a process trigger.
The caller can get the trigger value from any process in the system.
Please consult the SCIOPTA - Kernel, User’s Guide for more information about SCIOPTA trigger.

```c
sc_triggerval_t sc_triggerValueGet (sc_pid_t pid);
```

**Parameter**

*pid*

ID of the process which triggers needs to be returned.

**Return Value**

This system call returns the value of the trigger.
This system call is used to set the value of a process trigger to any positive value.
The caller can only set the trigger value of its own trigger.
Please consult the SCIOPTA - Kernel, User’s Guide for more information about SCIOPTA trigger.

```c
void sc_triggerValueSet ( sc_triggerval_t value );
```

**Parameter**

*value*  
The new trigger value which will be stored

**Return Value**

*none*
3 System Call Reference

3.81 sc_triggerWait

This system call is used to wait on the process trigger.

The `sc_triggerWait()` call will wait on the trigger of the callers process. The trigger value will be decremented by the value `dec` of the parameters.

If the trigger value becomes negative or equal zero, the calling process will be suspended and swapped out. The process will become ready again if the trigger value becomes positive. This occurs if another process has activated the trigger a sufficient number of times or with a sufficient trigger value.

The caller can also specify a time-out value `tmo`. The caller will not wait longer than the specified time for the trigger. If the time-out expires the process will be swapped in again.

Please consult the SCIOPTA - Kernel, User’s Guide for more information about SCIOPTA trigger.

```c
int sc_triggerWait(
    sc_triggerval_t dec,
    sc_ticks_t tmo
);
```

Parameter

`dec`

The number to decrease the process trigger value.

`tmo`

Maximum time-out value in system ticks which the process will wait on its trigger.

A value of `tmo == 0` will generate a system error.

Return Value

If the trigger was decremented but the calling process was not suspended as the trigger value is still positive or zero, the return value is zero.

If the trigger occurred and the trigger value became negative, the return value is one.

If the time-out expired, the return value is minus one.
4 Kernel Error Codes

4.1 Introduction

SCIOPTA uses a centralized mechanism for error reporting called Error Hooks.

There are two error hooks available:

A) Module Error Hook
B) Global Error Hook

If the kernel detect an error condition it will first call the module error hook and if it is not available call the global error hook. Error hooks are normal error handling functions and must be written by the user. Depending on the type of error (fatal or non-fatal) it will not be possible to return from an error hook. If there are no error hooks present the kernel will enter an infinite loop.

When the error hook is called from the kernel, all information about the error are transferred in 32-bit error word parameter.

<table>
<thead>
<tr>
<th>Function Code</th>
<th>Error Code</th>
<th>Error Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 Bits</td>
<td>12 Bits</td>
<td>12 Bits</td>
</tr>
</tbody>
</table>

32 Bits

Figure 4-1: 32-bit Error Word

The Function Code defines from what SCIOPTA system call the error was initiated. The Error Code contains the specific error information and the Error Type informs about the severeness of the error.

Please consult the SCIOPTA - Kernel, User’s Guide for more information about error hooks.

4.2 Include Files

The error codes are defined in the err.h include file.

The error descriptions are defined in the errtxt.h include file.
## 4.3 Function Codes

<table>
<thead>
<tr>
<th>Name</th>
<th>Number</th>
<th>Function Call</th>
<th>Error Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC.MSGALLOC</td>
<td>0x01</td>
<td>sc_msgAlloc</td>
<td></td>
</tr>
<tr>
<td>SC.MSGFREE</td>
<td>0x02</td>
<td>sc_msgFree</td>
<td></td>
</tr>
<tr>
<td>SC.MSGADDRGET</td>
<td>0x03</td>
<td>sc_msgAddrGet</td>
<td></td>
</tr>
<tr>
<td>SC.MSGSENDGET</td>
<td>0x04</td>
<td>sc_msgSendGet</td>
<td></td>
</tr>
<tr>
<td>SC.MSGSIZEGET</td>
<td>0x05</td>
<td>sc_msgSizeGet</td>
<td></td>
</tr>
<tr>
<td>SC.MSGOWNERGET</td>
<td>0x06</td>
<td>sc_msgOwnerGet</td>
<td></td>
</tr>
<tr>
<td>SC.MSGTX</td>
<td>0x07</td>
<td>sc_msgTx</td>
<td></td>
</tr>
<tr>
<td>SC.MSGTXALIAS</td>
<td>0x08</td>
<td>sc_msgTxAlias</td>
<td></td>
</tr>
<tr>
<td>SC.MSGRX</td>
<td>0x09</td>
<td>sc_msgRx</td>
<td></td>
</tr>
<tr>
<td>SC.MSGPOOLIDGET</td>
<td>0x0A</td>
<td>sc_msgPoolIdGet</td>
<td></td>
</tr>
<tr>
<td>SC.MSGACQUIRE</td>
<td>0x0B</td>
<td>sc_msgAcquire</td>
<td></td>
</tr>
<tr>
<td>SC_POOLCREATE</td>
<td>0x0C</td>
<td>sc_poolCreate</td>
<td></td>
</tr>
<tr>
<td>SC_POOLRESET</td>
<td>0x0D</td>
<td>sc_poolReset</td>
<td></td>
</tr>
<tr>
<td>SC_POOLKILL</td>
<td>0x0E</td>
<td>sc_poolKill</td>
<td></td>
</tr>
<tr>
<td>SC_POOLINFO</td>
<td>0x0F</td>
<td>sc_poolInfo</td>
<td></td>
</tr>
<tr>
<td>SC_POOLDEFAULT</td>
<td>0x10</td>
<td>sc_poolDefault</td>
<td></td>
</tr>
<tr>
<td>SC_POOLIDGET</td>
<td>0x11</td>
<td>sc_poolIdGet</td>
<td></td>
</tr>
<tr>
<td>SC_SYSPOOLKILL</td>
<td>0x12</td>
<td>sc_sysPoolKill</td>
<td></td>
</tr>
<tr>
<td>SC_PROCPRIOGET</td>
<td>0x13</td>
<td>sc_procPrioGet</td>
<td></td>
</tr>
<tr>
<td>SC_PROCPRIOSET</td>
<td>0x14</td>
<td>sc_procPrioSet</td>
<td></td>
</tr>
<tr>
<td>SC_PROCSLICEGET</td>
<td>0x15</td>
<td>sc_procSliceGet</td>
<td></td>
</tr>
<tr>
<td>SC_PROCSLICESET</td>
<td>0x16</td>
<td>sc_procSliceSet</td>
<td></td>
</tr>
<tr>
<td>SC_PROCIDGET</td>
<td>0x17</td>
<td>sc_procIdGet</td>
<td></td>
</tr>
<tr>
<td>SC_PROCPRIPIDGET</td>
<td>0x18</td>
<td>sc_procPpidGet</td>
<td></td>
</tr>
<tr>
<td>SC_PROCNAMEGET</td>
<td>0x19</td>
<td>sc_procNameGet</td>
<td></td>
</tr>
<tr>
<td>SC_PROCSCHEDULE</td>
<td>0x1A</td>
<td>sc_procStart</td>
<td></td>
</tr>
<tr>
<td>SC_PROCSTOP</td>
<td>0x1B</td>
<td>sc_procStop</td>
<td></td>
</tr>
<tr>
<td>SC_PROCVARINIT</td>
<td>0x1C</td>
<td>sc_procVarInit</td>
<td></td>
</tr>
<tr>
<td>SC_PROCSCHEDELOCK</td>
<td>0x1D</td>
<td>sc_procSchedUnlock</td>
<td></td>
</tr>
<tr>
<td>SC_PROCPRIOCREATESTATIC</td>
<td>0x1E</td>
<td>sc_procPrioCreateStatic</td>
<td></td>
</tr>
<tr>
<td>SCPROCINTCREATESTATIC</td>
<td>0x1F</td>
<td>sc_procIntCreateStatic</td>
<td></td>
</tr>
<tr>
<td>SC_PROCTIMCREATESTATIC</td>
<td>0x20</td>
<td>sc_procTimCreateStatic</td>
<td></td>
</tr>
<tr>
<td>SC_PROCUINTCREATESTATIC</td>
<td>0x21</td>
<td>sc_procUintCreateStatic</td>
<td></td>
</tr>
<tr>
<td>SC_PROCPRIOCREATE</td>
<td>0x22</td>
<td>sc_procPrioCreate</td>
<td></td>
</tr>
<tr>
<td>SC_PROCINTCREATE</td>
<td>0x23</td>
<td>sc_procIntCreate</td>
<td></td>
</tr>
<tr>
<td>SC_PROCTIMCREATE</td>
<td>0x24</td>
<td>sc_procTimCreate</td>
<td></td>
</tr>
<tr>
<td>SC_PROCUINTCREATE</td>
<td>0x25</td>
<td>sc_procUintCreate</td>
<td></td>
</tr>
<tr>
<td>SC_PROCSTOP</td>
<td>0x26</td>
<td>sc_procStop</td>
<td></td>
</tr>
<tr>
<td>SC_PROCVARINIT</td>
<td>0x27</td>
<td>sc_procVarInit</td>
<td></td>
</tr>
<tr>
<td>SC_PROCSCHEDELOCK</td>
<td>0x28</td>
<td>sc_procSchedUnlock</td>
<td></td>
</tr>
<tr>
<td>SC_PROCPRIOCREATESTATIC</td>
<td>0x29</td>
<td>sc_procPrioCreateStatic</td>
<td></td>
</tr>
<tr>
<td>SC_PROCINTCREATESTATIC</td>
<td>0x2A</td>
<td>sc_procIntCreateStatic</td>
<td></td>
</tr>
<tr>
<td>SC_PROCTIMCREATESTATIC</td>
<td>0x2B</td>
<td>sc_procTimCreateStatic</td>
<td></td>
</tr>
<tr>
<td>SC_PROCUINTCREATESTATIC</td>
<td>0x2C</td>
<td>sc_procUintCreateStatic</td>
<td></td>
</tr>
<tr>
<td>SC_PROCSTOP</td>
<td>0x2D</td>
<td>sc_procStop</td>
<td></td>
</tr>
<tr>
<td>SC_PROCVARINIT</td>
<td>0x2E</td>
<td>sc_procVarInit</td>
<td></td>
</tr>
<tr>
<td>SC_PROCSCHEDELOCK</td>
<td>0x2F</td>
<td>sc_procSchedUnlock</td>
<td></td>
</tr>
<tr>
<td>SC_PROCPRIOCREATESTATIC</td>
<td>0x30</td>
<td>sc_procPrioCreateStatic</td>
<td></td>
</tr>
<tr>
<td>SC_PROCINTCREATE</td>
<td>0x31</td>
<td>sc_procIntCreate</td>
<td></td>
</tr>
<tr>
<td>SC_PROCTIMCREATE</td>
<td>0x32</td>
<td>sc_procTimCreate</td>
<td></td>
</tr>
<tr>
<td>SC_PROCUINTCREATE</td>
<td>0x33</td>
<td>sc_procUintCreate</td>
<td></td>
</tr>
<tr>
<td>SC_PROCSTOP</td>
<td>0x34</td>
<td>sc_procStop</td>
<td></td>
</tr>
<tr>
<td>SC_PROCVARINIT</td>
<td>0x35</td>
<td>sc_procVarInit</td>
<td></td>
</tr>
<tr>
<td>SC_PROCSCHEDELOCK</td>
<td>0x36</td>
<td>sc_procSchedUnlock</td>
<td></td>
</tr>
<tr>
<td>SC_PROCPRIOCREATESTATIC</td>
<td>0x37</td>
<td>sc_procPrioCreateStatic</td>
<td></td>
</tr>
<tr>
<td>SC_PROCINTCREATESTATIC</td>
<td>0x38</td>
<td>sc_procIntCreateStatic</td>
<td></td>
</tr>
<tr>
<td>SC_PROCTIMCREATESTATIC</td>
<td>0x39</td>
<td>sc_procTimCreateStatic</td>
<td></td>
</tr>
<tr>
<td>SC_PROCUINTCREATESTATIC</td>
<td>0x3A</td>
<td>sc_procUintCreateStatic</td>
<td></td>
</tr>
</tbody>
</table>
## 4 Kernel Error Codes

<table>
<thead>
<tr>
<th>Name</th>
<th>Number</th>
<th>Function Call</th>
<th>Error Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC_PROCSCHEDLOCK</td>
<td>0x37</td>
<td>sc_procSchedLock</td>
<td></td>
</tr>
<tr>
<td>SC_PROCVARGET</td>
<td>0x38</td>
<td>sc_procVarGet</td>
<td></td>
</tr>
<tr>
<td>SC_PROCVARSET</td>
<td>0x39</td>
<td>sc_procVarSet</td>
<td></td>
</tr>
<tr>
<td>SC_PROCVARDEL</td>
<td>0x3a</td>
<td>sc_procVarDel</td>
<td></td>
</tr>
<tr>
<td>SC_PROCVARRM</td>
<td>0x3b</td>
<td>sc_procVarRm</td>
<td></td>
</tr>
<tr>
<td>SC_PROCUNOBSERVE</td>
<td>0x3c</td>
<td>sc_procUnobserve</td>
<td></td>
</tr>
<tr>
<td>SC_PROCSPATHGET</td>
<td>0x3d</td>
<td>sc_procPathGet</td>
<td></td>
</tr>
<tr>
<td>SC_MODULECREATE</td>
<td>0x40</td>
<td>sc_moduleCreate</td>
<td></td>
</tr>
<tr>
<td>SC_MODULEKILL</td>
<td>0x41</td>
<td>sc_moduleKill</td>
<td></td>
</tr>
<tr>
<td>SC_MODULENAMEGET</td>
<td>0x42</td>
<td>sc_moduleNameGet</td>
<td></td>
</tr>
<tr>
<td>SC_MODULEIDGET</td>
<td>0x43</td>
<td>sc_moduleIdGet</td>
<td></td>
</tr>
<tr>
<td>SC_MODULEINFO</td>
<td>0x44</td>
<td>sc_moduleInfo</td>
<td></td>
</tr>
<tr>
<td>SC_MODULEPRIOSET</td>
<td>0x45</td>
<td>sc_modulePrioSet</td>
<td></td>
</tr>
<tr>
<td>SC_MODULEPRIOGET</td>
<td>0x46</td>
<td>sc_modulePrioGet</td>
<td></td>
</tr>
<tr>
<td>SC_MODULEFRIENDADD</td>
<td>0x47</td>
<td>sc_moduleFriendAdd</td>
<td></td>
</tr>
<tr>
<td>SC_MODULEFRIENDRM</td>
<td>0x48</td>
<td>sc_moduleFriendRm</td>
<td></td>
</tr>
<tr>
<td>SC_MODULEFRIENDGET</td>
<td>0x49</td>
<td>sc_moduleFriendGet</td>
<td></td>
</tr>
<tr>
<td>SC_MODULEFRIENDNON</td>
<td>0x4A</td>
<td>sc_moduleFriendNone</td>
<td></td>
</tr>
<tr>
<td>SC_MODULEFRIENDALL</td>
<td>0x4B</td>
<td>sc_moduleFriendAll</td>
<td></td>
</tr>
<tr>
<td>SC_TRIGGERVALUESET</td>
<td>0x50</td>
<td>sc_triggerValueSet</td>
<td></td>
</tr>
<tr>
<td>SC_TRIGGERVALUEGET</td>
<td>0x51</td>
<td>sc_triggerValueGet</td>
<td></td>
</tr>
<tr>
<td>SC_TRIGGER</td>
<td>0x52</td>
<td>sc_trigger</td>
<td></td>
</tr>
<tr>
<td>SC_TRIGGERWAIT</td>
<td>0x53</td>
<td>sc_triggerWait</td>
<td></td>
</tr>
<tr>
<td>SC_TMOADD</td>
<td>0x58</td>
<td>sc_tmoAdd</td>
<td></td>
</tr>
<tr>
<td>SC_TMO</td>
<td>0x59</td>
<td>sc_tmo</td>
<td></td>
</tr>
<tr>
<td>SC_SLEEP</td>
<td>0x5A</td>
<td>sc_sleep</td>
<td></td>
</tr>
<tr>
<td>SC_TMORM</td>
<td>0x5B</td>
<td>sc_tmorm</td>
<td></td>
</tr>
<tr>
<td>SC_CONNECTORREGISTER</td>
<td>0x60</td>
<td>sc_connectorRegister</td>
<td></td>
</tr>
<tr>
<td>SC_CONNECTORUNREGISTER</td>
<td>0x61</td>
<td>sc_connectorUnregister</td>
<td></td>
</tr>
<tr>
<td>SC_DISPATCHER</td>
<td>0x62</td>
<td>dispatcher</td>
<td></td>
</tr>
</tbody>
</table>
# 4 Kernel Error Codes

## 4.4 Error Codes

<table>
<thead>
<tr>
<th>Name</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KERNEL_EILL_POOL_ID</td>
<td>0x001</td>
<td>Illegal pool ID.</td>
</tr>
<tr>
<td>KERNEL_ENO_MOORE_POOL</td>
<td>0x002</td>
<td>No more pool.</td>
</tr>
<tr>
<td>KERNEL_EILL_POOL_SIZE</td>
<td>0x003</td>
<td>Illegal pool size.</td>
</tr>
<tr>
<td>KERNEL_EPOOL_IN_USE</td>
<td>0x004</td>
<td>Pool still in use.</td>
</tr>
<tr>
<td>KERNEL_EILL_NUM_SIZES</td>
<td>0x005</td>
<td>Illegal number of buffer sizes.</td>
</tr>
<tr>
<td>KERNEL_EILL_BUF_SIZES</td>
<td>0x006</td>
<td>Illegal buffersizes.</td>
</tr>
<tr>
<td>KERNEL_EILL_BUFSIZE</td>
<td>0x007</td>
<td>Illegal buffersize.</td>
</tr>
<tr>
<td>KERNEL_EOUTSIDE_POOL</td>
<td>0x008</td>
<td>Message outside pool.</td>
</tr>
<tr>
<td>KERNEL_EMSG_HD_CORRUPT</td>
<td>0x009</td>
<td>Message header corrupted.</td>
</tr>
<tr>
<td>KERNEL_ENIL_PTR</td>
<td>0x00A</td>
<td>NIL pointer.</td>
</tr>
<tr>
<td>KERNEL_EENLARGE_MSG</td>
<td>0x00B</td>
<td>Message enlarged.</td>
</tr>
<tr>
<td>KERNEL_ENOT_OWNER</td>
<td>0x00C</td>
<td>Not owner of the message.</td>
</tr>
<tr>
<td>KERNEL_EOUT_OF_MEMORY</td>
<td>0x00D</td>
<td>Out of memory.</td>
</tr>
<tr>
<td>KERNEL_EILL_VECTOR</td>
<td>0x00E</td>
<td>Illegal interrupt vector.</td>
</tr>
<tr>
<td>KERNEL_EILL_SLICE</td>
<td>0x00F</td>
<td>Illegal time slice.</td>
</tr>
<tr>
<td>KERNEL_ENO_KERNELD</td>
<td>0x010</td>
<td>No kernel daemon started.</td>
</tr>
<tr>
<td>KERNEL_EMSG_ENDMARK_CORRUPT</td>
<td>0x011</td>
<td>Message endmark corrupted.</td>
</tr>
<tr>
<td>KERNEL_EMSG_PREV_ENDMARK_CORRUPT</td>
<td>0x012</td>
<td>Previous message’s endmark corrupted.</td>
</tr>
<tr>
<td>KERNEL_EILL_DEFPOOL_ID</td>
<td>0x013</td>
<td>Illegal default pool ID.</td>
</tr>
<tr>
<td>KERNEL_ELOCKED</td>
<td>0x014</td>
<td>Illegal system call while scheduler locked.</td>
</tr>
<tr>
<td>KERNEL_EILL_PROCTYPE</td>
<td>0x015</td>
<td>Illegal process type.</td>
</tr>
<tr>
<td>KERNEL_EILL_INTERRUPT</td>
<td>0x016</td>
<td>Illegal interrupt.</td>
</tr>
<tr>
<td>KERNEL_EUNLOCK_WO_LOCK</td>
<td>0x01F</td>
<td>Unlock without lock.</td>
</tr>
<tr>
<td>KERNEL_EILL_PID</td>
<td>0x020</td>
<td>Illegal process ID.</td>
</tr>
<tr>
<td>KERNEL_ENO_MORE_PROC</td>
<td>0x021</td>
<td>No more processes.</td>
</tr>
<tr>
<td>KERNEL_EMODULE_TOO_SMALL</td>
<td>0x022</td>
<td>Module size too small.</td>
</tr>
<tr>
<td>KERNEL_ESTART_NOT_STOPPED</td>
<td>0x023</td>
<td>Starting of a not stopped process.</td>
</tr>
<tr>
<td>KERNEL_EILL_PROC</td>
<td>0x024</td>
<td>Illegal process.</td>
</tr>
<tr>
<td>KERNEL_EILL_NAME</td>
<td>0x025</td>
<td>Illegal name.</td>
</tr>
<tr>
<td>KERNEL_EILL_TARGET_NAME</td>
<td>0x025</td>
<td>Illegal target name.</td>
</tr>
<tr>
<td>KERNEL_EILL_MODULE_NAME</td>
<td>0x025</td>
<td>Illegal module name.</td>
</tr>
<tr>
<td>KERNEL_EILL_MODID</td>
<td>0x027</td>
<td>Illegal module ID.</td>
</tr>
<tr>
<td>KERNEL_EILL_PRIORITY</td>
<td>0x028</td>
<td>Illegal priority.</td>
</tr>
<tr>
<td>KERNEL_EILL_STACKSIZE</td>
<td>0x029</td>
<td>Illegal stacksize.</td>
</tr>
<tr>
<td>KERNEL_ENO_MORE_MODULE</td>
<td>0x02A</td>
<td>No more modules available.</td>
</tr>
<tr>
<td>KERNEL_EILL_PARAMETER</td>
<td>0x02B</td>
<td>Illegal parameter.</td>
</tr>
<tr>
<td>KERNEL_EILL_PROC_NAME</td>
<td>0x025</td>
<td>Illegal process name.</td>
</tr>
<tr>
<td>KERNEL_EPROC_NOT_PRIO</td>
<td>0x02D</td>
<td>Not a prioritized process.</td>
</tr>
<tr>
<td>KERNEL_ESTACK_OVERFLOW</td>
<td>0x02E</td>
<td>Stack overflow.</td>
</tr>
<tr>
<td>KERNEL_ESTACK_UNDERFLOW</td>
<td>0x02F</td>
<td>Stack underflow.</td>
</tr>
<tr>
<td>KERNEL_EILL_VALUE</td>
<td>0x030</td>
<td>Illegal value.</td>
</tr>
<tr>
<td>KERNEL_EALREADY_DEFINED</td>
<td>0x031</td>
<td>Already defined.</td>
</tr>
<tr>
<td>KERNEL_ENO_MORE_CONNECTOR</td>
<td>0x032</td>
<td>No more connectors available.</td>
</tr>
<tr>
<td>KERNEL_EPROC_TERMINATE</td>
<td>0xFF</td>
<td>Process terminated.</td>
</tr>
</tbody>
</table>
### 4.5 Error Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC_ERR_TARGET_FATAL</td>
<td>0x01</td>
<td>This type of error will stop the whole target.</td>
</tr>
<tr>
<td>SC_ERR_MODULE_FATAL</td>
<td>0x02</td>
<td>This type of error results in killing the module if an error hook returns a value of !=0.</td>
</tr>
<tr>
<td>SC_ERR_PROCESS_FATAL</td>
<td>0x04</td>
<td>This type of error results in killing the process if an error hook returns a value of !=0.</td>
</tr>
<tr>
<td>SC_ERR_TARGET_WARNING</td>
<td>0x10</td>
<td>Warning on target level. The system continues if an error hook is installed.</td>
</tr>
<tr>
<td>SC_ERR_MODULE_WARNING</td>
<td>0x20</td>
<td>Warning on module level. The system continues if an error hook is installed.</td>
</tr>
<tr>
<td>SC_ERR_PROC_WARNING</td>
<td>0x40</td>
<td>Warning on process level. The system continues if an error hook is installed.</td>
</tr>
</tbody>
</table>
5 Manual Revision

5.1 Manual Version 1.6

- Chapter 3.27 sc_msgRx, \texttt{tmo} parameter, SC\_TMO\_NONE replaced by SC\_NO\_TMO. Parameter better specified.
- Chapter 3.27 sc_msgRx, \texttt{wanted} parameter, NULL replaced by SC\_MSGRX\_ALL.
- Chapter 3.7 sc\_miscError, \texttt{err} parameter, bits 0, 1 and 2 documented.
- Chapter 3.44 sc\_procIdGet, if parameter \texttt{path} is NULL and parameter \texttt{tmo} is SC\_NO\_TMO this system call returns the callers process ID.

5.2 Manual Version 1.5

- All \texttt{union sc\_msg *} changed to \texttt{sc\_msg\_t} to support SCIOPTA 16 Bit systems (NEAR pointer).
- All \texttt{union sc\_msg **} changed to \texttt{sc\_msgptr\_t} to support SCIOPTA 16 Bit systems (NEAR pointer).
- Chapter 6, System Call Reference, page layout for all system calls modified.
- Chapter 6.81 sc\_triggerWait, third paragraph rewritten.
- Chapters 6.9 sc\_moduleCreate and 6.17 sc\_moduleKill, information about \texttt{sc\_kerneld} are given.
- Chapter 6.44 sc\_procIdGet, added text: this parameter is not allowed if asynchronous timeout is disabled at system configuration (\texttt{sconf}).

5.3 Manual Version 1.4

- Chapter 4.7.3.2 Example, OS\_INT\_PROCESS changed into correct SC\_INT\_PROCESS.
- Chapter 2.3.4.4 Init Process, rewritten.
- Chapter 4.5 Processes, former chapters 4.5.6 Idle Process and 4.5.7 Supervisor Process removed.
- Chapter 4.5.1 Introduction, last paragraph about supervisor processes added.
- Chapter 4.5.5 Init Process, rewritten.
- Chapter 6.8 sc\_miscErrorHookRegister, syntax corrected.
- Chapter 6.21 sc\_mscAlloc, time-out parameter \texttt{tmo} better specified.
- Chapter 6.27 sc\_msgRx, time-out parameter \texttt{tmo} better specified.
- Chapter 4.10.4 Error Hook Declaration Syntax, user \(!=0\) user error.
- Chapter 4.9 SCIOPTA Daemons, moved from chapter 2.9 and rewritten.
- Chapter 6.41 sc\_procDaemonRegister, last paragraph of the description rewritten.
- Chapters 6.45 sc\_procIntCreate, 6.46 sc\_procKill, 6.51 sc\_procPrioCreate, 6.60 sc\_procTimCreate and 6.62 sc\_procUsrIntCreate, information about \texttt{sc\_kerneld} are given.
- Chapter 4.10.5 Example, added.
5 Manual Revision

5.4 Manual Version 1.3

- Chapter 6.26 sc_msgPoolIdGet, return value SC_DEFAULT_POOL defined.
- Chapter 6.33 sc_poolCreate, pool size formula added.
- Chapter 2.4.4 Message Pool, maximum number of pools for compact kernel added.
- Chapter 4.8 SCIOPTA Memory Manager - Message Pools, added.
- Chapter 6.9 sc_moduleCreate, module size calculation modified.
- Chapter 6.40 sc_procCreate, 6.45 sc_procIntCreate, 6.51 sc_procPrioCreate and 6.60 sc_procTim Create, stacksize calculation modified.
5 Manual Revision

5.5 Manual Version 1.2

- Top cover back side: Address of SCIOPTA France added.
- Chapter 2.6 Trigger, second paragraph: At process creation the value of the trigger is initialized to one.
- Chapter 2.6 Trigger, third paragraph: The sc_triggerWait() call decrements the value of the trigger and the calling process will be blocked and swapped out if the value gets negative or equal zero.
- Chapter 2.7 Process Variables, second paragraph: The tag and the process variable have a fixed size large enough to hold a pointer.
- Chapter 2.7 Process Variables, third paragraph: Last sentence rewritten.
- Chapter 4.5.3.1 Interrupt Process Declaration Syntax, irg_src is of type int added.
- Chapter 4.5.6 Idle Process, added.
- Chapter 4.10.4 Error Hook Declaration Syntax, Parameter user : user != 0 (User error).
- System call sc_procRegisterDaemon changed to sc_DaemonRegister and sc_procUnregisterDaemon changed to sc_procDaemonUnregister.
- System call sc_m miscErrorHookRegister, return values better specified.
- System call sc_moduleCreate, parameter size value “code” added in Formula.
- System call sc_moduleNameGet, return value NULL added.
- System call sc_msgAcquire, condition modified.
- System Call sc_msgAlloc, SC_DEFAULT_POOL better specified.
- SystemCall sc_msgHookRegister, description modified and return value better specified.
- System call sc_msgRx, parameters better specified.
- System call sc_poolHookRegister, return value better specified.
- System call sc_procHookRegister, return value better specified.
- System call sc_proclDget, last paragraph in Description added.
- System calls sc_procVarDel, sc_procVarGet and procVarSet, return value !=0 introduced.
- Chapter 7.3 Function Codes, errors 0x38 to 0x3d added.
- System call sc_procUnobserve added.
- Chapters 2.5.2 System Module and 4.3 Modules, the following sentence was removed: The system module runs always on supervisor level and has all access rights.
- Chapter 2.5.3 Messages and Modules, third paragraph rewritten.
- Chapter 6.31 sc_msgTx, fifth paragraph rewritten.
5 Manual Revision

5.6 Manual Version 1.1

- System call `sc_moduleInfo` has now a return parameter.
- New system call `sc_procPathGet`.
- System call `sc_moduleCreate` formula to calculate the size of the module (parameter `size`) added.
- Chapter 4.12 SCIOPTA Design Rules, moved at the end of chapter “System Design”.
- New chapter 4.6 Addressing Processes.
- Chapter 7 Kernel Error Codes, new sequence of sub chapters. Smaller font used.
- Chapter 4.10 Error Hook, completely rewritten.
- New chapter 4.11 System Start.

5.7 Manual Version 1.0

Initial version.
## Index

### A
- Addressee .......................................................................................................................................................... 3-22
- Allocate ........................................................................................................................................................... 3-23
- Allocation timing parameter .......................................................................................................................... 3-24

### B
- BSP ........................................................................................................................................................... 1-1
- Buffer ........................................................................................................................................................... 3-23
- Buffer sizes .................................................................................................................................................. 3-38

### C
- CONNNECTOR ............................................................................................................................................... 3-1, 3-2
- Connector process ........................................................................................................................................ 3-2, 3-3
- Connector process calls ............................................................................................................................. 2-6
- CRC .......................................................................................................................................................... 3-3, 3-4
- Create ...................................................................................................................................................... 3-60
- Creator ....................................................................................................................................................... 3-59

### D
- Default pool ................................................................................................................................................ 3-39
- DRUID ....................................................................................................................................................... 1-1

### E
- Effective priority ........................................................................................................................................ 3-9
- err.h ........................................................................................................................................................... 4-1
- errno ........................................................................................................................................................... 3-5, 3-6
- Error code ............................................................................................................................................... 4-4, 4-5
- Error hook ................................................................................................................................................ 3-7, 4-1
- Error number ........................................................................................................................................ 3-5, 3-6
- Error type ............................................................................................................................................... 4-4, 4-5
- errtxt.h ................................................................................................................................................... 4-1
- External processes .................................................................................................................................... 3-1

### F
- File system ................................................................................................................................................. 1-1
- Free-list .................................................................................................................................................. 3-44
- Friend ...................................................................................................................................................... 3-12, 3-13, 3-14, 3-16
- Friend modules ..................................................................................................................................... 3-12, 3-13, 3-14, 3-15, 3-16
- Friend set ............................................................................................................................................. 3-15
- Function calls ...................................................................................................................................... 1-3
- Function code ......................................................................................................................................... 4-1, 4-2

### G
- Global error hook ..................................................................................................................................... 4-1
- Global message hook ............................................................................................................................... 3-27
- Global variables ..................................................................................................................................... 1-3
I
ID of a message pool ................................................................. 3-41
Include files .............................................................................. 4-1
Input/Output management .......................................................... 1-3
Installation information ............................................................. 1-1
Internet protocols ....................................................................... 1-1
Interprocess communication ....................................................... 1-2, 1-3
Interrupt process ........................................................................ 3-9, 3-45, 3-53
Interrupt vector ........................................................................... 3-80
IPS ................................................................................................. 1-1

K
Kernel error codes ....................................................................... 4-1
Kernel tick counter ....................................................................... 3-83
Kernel tick function ..................................................................... 3-83
KERNEL_EALREADY_DEFINED ...................................................... 4-4
KERNEL_EENLARGE_MSG ............................................................. 4-4
KERNEL_FILL_BUF_SIZES ............................................................... 4-4
KERNEL_FILL_BUFSIZE ................................................................. 4-4
KERNEL_FILL_DEFPOOL_ID ............................................................ 4-4
KERNEL_FILL_DEFPROC ................................................................. 4-4
KERNEL_FILL_INTERRUPT ............................................................... 4-4
KERNEL_FILL_MODULE ................................................................. 4-4
KERNEL_FILL_MODULE_NAME ......................................................... 4-4
KERNEL_FILL_NAME ............................................................... 4-4
KERNEL_FILL_NUM_SIZES .............................................................. 4-4
KERNEL_FILL_PARAMETER ............................................................. 4-4
KERNEL_FILL_PID ................................................................. 4-4
KERNEL_FILL_POOL_ID ................................................................. 4-4
KERNEL_FILL_POOL_SIZE ............................................................ 4-4
KERNEL_FILL_PRIORITY ............................................................... 4-4
KERNEL_FILL_PROC ................................................................. 4-4
KERNEL_FILL_PROC_NAME ........................................................... 4-4
KERNEL_FILL_PROCTYPE ............................................................. 4-4
KERNEL_FILL_SLICE ................................................................. 4-4
KERNEL_FILL_STACKSIZE ............................................................. 4-4
KERNEL_FILL_TARGET_NAME ......................................................... 4-4
KERNEL_FILL_VALUE ................................................................. 4-4
KERNEL_FILL_VECTOR ................................................................. 4-4
KERNEL_ELOCKED ........................................................................ 4-4
KERNEL_EMODULE_TOO_SMALL ..................................................... 4-4
KERNEL_EMSG_ENDMARK_CORRUPT ........................................... 4-4
KERNEL_EMSG_HDR_CORRUPT ....................................................... 4-4
KERNEL_EMSG_PREV_ENDMARK_CORRUPT .................................... 4-4
KERNEL_ENIL_PTR ................................................................. 4-4
KERNEL_ENO_KERNELD ............................................................... 4-4
KERNEL_ENO_MORE_POOL ............................................................ 4-4
KERNEL_ENO_MORE_PROC ........................................................... 4-4
KERNEL_ENO_MORE_CONNECTOR ................................................... 4-4
KERNEL_ENO_MORE_MODULE ........................................................ 4-4
KERNEL_ENO_OWNER ................................................................. 4-4
KERNEL_EOUT_OF_MEMORY .......................................................... 4-4
<table>
<thead>
<tr>
<th>Index</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Locked</td>
<td>3-64</td>
</tr>
<tr>
<td>Manual revision</td>
<td>5-1</td>
</tr>
<tr>
<td>Manual version 1.0</td>
<td>5-4</td>
</tr>
<tr>
<td>Manual version 1.1</td>
<td>5-4</td>
</tr>
<tr>
<td>Manual version 1.2</td>
<td>5-3</td>
</tr>
<tr>
<td>Manual version 1.3</td>
<td>5-2</td>
</tr>
<tr>
<td>Manual version 1.4</td>
<td>5-1</td>
</tr>
<tr>
<td>mcb</td>
<td>3-18</td>
</tr>
<tr>
<td>Memory buffer</td>
<td>3-23</td>
</tr>
<tr>
<td>Memory management unit</td>
<td>1-1</td>
</tr>
<tr>
<td>Message ID</td>
<td>3-23</td>
</tr>
<tr>
<td>Message pool</td>
<td>3-37</td>
</tr>
<tr>
<td>Message pool calls</td>
<td>2-4</td>
</tr>
<tr>
<td>Message queue</td>
<td>3-30</td>
</tr>
<tr>
<td>Message system calls</td>
<td>2-1</td>
</tr>
<tr>
<td>Miscellaneous and error calls</td>
<td>2-6</td>
</tr>
<tr>
<td>Module control block</td>
<td>3-18</td>
</tr>
<tr>
<td>Module error hook</td>
<td>4-1</td>
</tr>
<tr>
<td>Module message hook</td>
<td>3-27</td>
</tr>
<tr>
<td>Module priority</td>
<td>3-9</td>
</tr>
<tr>
<td>Module protection</td>
<td>3-9</td>
</tr>
<tr>
<td>Module system calls</td>
<td>2-3</td>
</tr>
<tr>
<td>module.h</td>
<td>3-18</td>
</tr>
<tr>
<td>Name of a module</td>
<td>3-20</td>
</tr>
<tr>
<td>Name of a process</td>
<td>3-56</td>
</tr>
<tr>
<td>Observe message</td>
<td>3-6</td>
</tr>
<tr>
<td>On-chip timer</td>
<td>3-83</td>
</tr>
<tr>
<td>Owner</td>
<td>3-28</td>
</tr>
<tr>
<td>Parent</td>
<td>3-59</td>
</tr>
<tr>
<td>Path</td>
<td>3-41, 3-51</td>
</tr>
</tbody>
</table>
6 Index

Path of a process ............................................................................................................................................ 3-58
pcb .............................................................................................................................................................. 3-18
plid ............................................................................................................................................................ 3-23
Pool control block .................................................................................................................................... 3-42
Pool create hook ..................................................................................................................................... 3-40
Pool ID ...................................................................................................................................................... 3-40
pool ID ....................................................................................................................................................... 3-29
Pool kill hook ........................................................................................................................................... 3-40
Prioritized process .................................................................................................................................. 3-45, 3-60, 3-62
Priority ......................................................................................................................................................... 3-62, 3-63, 3-65, 3-81
Process create hook .................................................................................................................................. 3-50
Process daemon ....................................................................................................................................... 3-48, 3-49
Process descriptor block .......................................................................................................................... 3-45
Process hook ........................................................................................................................................... 3-8, 3-50
Process ID .................................................................................................................................................. 3-51
Process kill hook ...................................................................................................................................... 3-50
Process swap hook ................................................................................................................................... 3-50
Process system calls .................................................................................................................................. 2-2
Process trigger ......................................................................................................................................... 3-90, 3-91, 3-92, 3-93
Process trigger calls .................................................................................................................................. 2-5
Process variable ....................................................................................................................................... 3-75, 3-76, 3-77, 3-78, 3-79
Process variable calls ............................................................................................................................... 2-6

R
Receive ......................................................................................................................................................... 3-30
Release notes ............................................................................................................................................. 1-1
Remote CPU ............................................................................................................................................ 3-55
Requested size .......................................................................................................................................... 3-34
Resource management ............................................................................................................................ 1-2
Return a message ..................................................................................................................................... 3-26

S
SC_CONNECTORREGISTER .......................................................................................................................... 4-3
sc_connectorRegister ............................................................................................................................... 2-6, 3-1
SC_CONNECTORUNREGISTER .................................................................................................................. 4-3
sc_connectorUnregister ............................................................................................................................ 2-6, 3-2
SC_DEFAULT_POOL .................................................................................................................................. 3-39
SC_DISPATCHER ....................................................................................................................................... 4-3
SC_ENDLESS_TMO .................................................................................................................................... 3-24
SC_ERR_MODULE_FATAL ............................................................................................................................ 4-5
SC_ERR_MODULE_WARNING ....................................................................................................................... 4-5
SC_ERR_PROC_WARNING ............................................................................................................................ 4-5
SC_ERR_PROCESS_FATAL ......................................................................................................................... 4-5
SC_ERR_TARGET_FATAL .............................................................................................................................. 4-5
SC_ERR_TARGET_WARNING ....................................................................................................................... 4-5
SC_FATAL_IF_TMO ..................................................................................................................................... 3-24
SC_ILLEGAL_PID ..................................................................................................................................... 3-52
SC_ILLEGAL_POOLID ................................................................................................................................. 3-41
sc_kerneld .................................................................................................................................................. 3-9, 3-19, 3-53, 3-55, 3-60, 3-70, 3-73
sc_mscrc .................................................................................................................................................... 2-6, 3-3
sc_mscrcContd ............................................................................................................................................ 2-6, 3-4
6 Index

sc_misckenErrnoGet .......................................................... 2-6, 3-5
sc_misckenErrnoSet .......................................................... 2-6, 3-6
sc_misckenError .............................................................. 2-6, 3-7
sc_misckenErrorHookRegister ........................................... 2-6, 3-8
SC_MODULECREATE .......................................................... 4-3
sc_moduleCreate ............................................................. 2-3, 3-9
SC_MODULEFRIENDADD .................................................... 4-3
sc_moduleFriendAdd ....................................................... 2-3, 3-12
SC_MODULEFRIENDALL .................................................... 4-3
sc_moduleFriendAll ....................................................... 2-3, 3-13
SC_MODULEFRIENDGET .................................................... 4-3
sc_moduleFriendGet ....................................................... 2-4, 3-14
SC_MODULEFRIENDNON ..................................................... 4-3
sc_moduleFriendNone ..................................................... 2-4, 3-15
SC_MODULEFRIENDRM ..................................................... 4-3
sc_moduleFriendRm ....................................................... 2-4, 3-16
SC_MODULEIDGET ............................................................. 4-3
sc_moduleIdGet ............................................................. 2-4, 3-17
SC_MODULEINFO .............................................................. 4-3
sc_moduleInfo ............................................................... 2-4, 3-18
SC_MODULEKILL ............................................................... 4-3
sc_moduleKill ............................................................... 2-4, 3-19
SC_MODULEKILL_KILL ........................................................ 3-19
SC_MODULENAMEGET ........................................................ 4-3
sc_moduleNameGet ........................................................ 2-4, 3-20
SC_MODULEPRIGET .......................................................... 4-3
SC_MODULEPROISET ........................................................ 4-3
SC_MSGACQUIRE ............................................................ 4-2
sc_msgAcquire .............................................................. 2-1, 3-21
SC_MSGADDRGET ............................................................ 4-2
sc_msgAddrGet .............................................................. 2-1, 3-22
SC_MSGALLOC ............................................................... 4-2
sc_msgAlloc ................................................................. 2-1, 3-23
sc_msgAllocOpt ............................................................. 2-1, 3-25
SC_MSGFREE ................................................................. 4-2
sc_msgFree ................................................................. 2-1, 3-26
sc_msgHookRegister ...................................................... 2-1, 3-27
SC_MGSOWNERGET ........................................................ 4-2
sc_mgsOwnerGet ........................................................... 2-1, 3-28
SC_MSGPOOLIDGET ........................................................ 4-2
sc_msgPoolIdGet ........................................................... 2-1, 3-29
SC_MSGRX ..................................................................... 4-2
sc_msgRx ................................................................. 2-1, 3-30
SC_MSGRX_BOTH ........................................................... 3-31
SC_MSGRX_MSGID .......................................................... 3-31
SC_MSGRX_PID .............................................................. 3-31
SC_MSGSIZEGET ............................................................ 4-2
sc_msgSizeGet .............................................................. 2-1, 3-33
SC_MSGSIZESET ............................................................ 4-2
sc_msgSizeSet .............................................................. 2-1, 3-34
SC_MSGSENDGET ........................................................... 4-2
sc_msgSendGet ............................................................. 2-1, 3-35
sc_msgSnadGet ......................................................................................................................... 2-1, 3-32
SC_MSGTX ........................................................................................................................................... 4-2
sc_msgTx .................................................................................................................................................... 2-1, 3-35
SC_MSGTXALIAS .................................................................................................................................. 4-2
sc_msgTxAlias ........................................................................................................................................ 2-1, 3-36
SC_NO_TMO ......................................................................................................................................... 3-24
SC_NOSUCH_MODULE .............................................................................................................................. 3-17
SC_PDB_INT ........................................................................................................................................ 3-46
SC_PDB_P prio ...................................................................................................................................... 3-46
SC_PDB_STATE_RUN .............................................................................................................................. 2-2, 3-49
SC_PDB_STATE_STP ....................................................................................................................................... 3-47
SC_PDB_TIM ........................................................................................................................................ 3-46
SC_POOLCREATE ................................................................................................................................... 4-2
sc_poolCreate ........................................................................................................................................ 2-4, 3-37
SC_POOLDEFAULT ................................................................................................................................... 4-2
sc_poolDefault ....................................................................................................................................... 2-4, 3-39
sc_poolHookRegister .......................................................................................................................... 2-4, 3-40
SC_POOLIDGET ....................................................................................................................................... 4-2
sc_poolIdGet .......................................................................................................................................... 2-4, 3-41
SC_POOLINFO ....................................................................................................................................... 4-2
sc_poolInfo ............................................................................................................................................ 2-4, 3-42
SC_POOLKILL ......................................................................................................................................... 4-2
sc_poolKill ............................................................................................................................................ 2-4, 3-43
SC_POOLRESET ..................................................................................................................................... 4-2
sc_poolReset ......................................................................................................................................... 2-4, 3-44
SC_PROC_CURRENT .................................................................................................................................. 3-59, 3-62, 3-66, 3-67
sc_procCreate ...................................................................................................................................... 2-2, 3-45
sc_procd ............................................................................................................................................... 3-48
sc_procDaemonRegister ....................................................................................................................... 2-2, 3-48
sc_proc DaemonUnregister .................................................................................................................. 2-2, 3-49
sc_procHookRegister .......................................................................................................................... 2-2, 3-50
SC_PROCIDGET ..................................................................................................................................... 4-2
sc_procIdGet .......................................................................................................................................... 2-2, 3-51
SC_PROCINTCREATE .............................................................................................................................. 4-2
sc_procIntCreate ................................................................................................................................... 2-2, 3-45, 3-53
SC_PROCINTCREATESTATIC ................................................................................................................ 4-2
SC_PROC KILL ....................................................................................................................................... 4-2
sc procKill ............................................................................................................................................. 2-2, 3-55
SC_PROC KILL_KILL .................................................................................................................................. 3-55
SC_PROCNAMEGET ............................................................................................................................... 4-2
sc_procNameGet .................................................................................................................................... 2-2, 3-56
SC_PROCNAMEGETMSG_REPLY ................................................................................................................ 3-56, 3-58
SC_PROC OBSERVE ............................................................................................................................. 4-2
sc procObserve ...................................................................................................................................... 2-2, 3-57
SC_PROC PATHGET ................................................................................................................................. 4-3
sc_procPathGet ...................................................................................................................................... 2-2, 3-58
SC_PROC PPIDGET .................................................................................................................................. 4-2
sc_procPpidGet ....................................................................................................................................... 2-2, 3-59
SC_PROC PRIOCREATE .......................................................................................................................... 4-2
sc.procPrioCreate ................................................................................................................................ 2-2
SC_PROC PRIOCREATESTATIC ................................................................................................................... 4-2
SC_PROC PRIORITYGET ........................................................................................................................... 4-2
<table>
<thead>
<tr>
<th>Function</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sc.procPrioGet</code></td>
<td>2-2, 3-62</td>
</tr>
<tr>
<td><code>SC_PROCPRIOSET</code></td>
<td>4-2</td>
</tr>
<tr>
<td><code>sc.procPrioSet</code></td>
<td>2-2, 3-63</td>
</tr>
<tr>
<td><code>SC_PROCSCHEDLOCK</code></td>
<td>4-3</td>
</tr>
<tr>
<td><code>sc.procSchedLock</code></td>
<td>2-2, 3-64</td>
</tr>
<tr>
<td><code>SC_PROCSCHEDUNLOCK</code></td>
<td>4-2</td>
</tr>
<tr>
<td><code>sc.procSchedUnLock</code></td>
<td>2-2, 3-65</td>
</tr>
<tr>
<td><code>SC_PROCSLICEGET</code></td>
<td>4-2</td>
</tr>
<tr>
<td><code>sc.procSliceGet</code></td>
<td>2-3, 3-66</td>
</tr>
<tr>
<td><code>SC_PROCSLICESET</code></td>
<td>4-2</td>
</tr>
<tr>
<td><code>sc.procSliceSet</code></td>
<td>2-3, 3-67</td>
</tr>
<tr>
<td><code>SC_PROCSTART</code></td>
<td>4-2</td>
</tr>
<tr>
<td><code>sc.procStart</code></td>
<td>2-3, 3-68</td>
</tr>
<tr>
<td><code>SC_PROCSTOP</code></td>
<td>4-2</td>
</tr>
<tr>
<td><code>sc.procStop</code></td>
<td>2-3, 3-69</td>
</tr>
<tr>
<td><code>SC_PROCTIMCREATE</code></td>
<td>4-2</td>
</tr>
<tr>
<td><code>sc.procTimCreate</code></td>
<td>2-3, 3-70</td>
</tr>
<tr>
<td><code>SC_PROCTIMCREATESTATIC</code></td>
<td>4-2</td>
</tr>
<tr>
<td><code>SC_PROCUNOBSERVE</code></td>
<td>4-3</td>
</tr>
<tr>
<td><code>sc.procUnobserve</code></td>
<td>2-3, 3-72</td>
</tr>
<tr>
<td><code>SC_PROCUSRINTCREATE</code></td>
<td>4-2</td>
</tr>
<tr>
<td><code>sc.procUsrIntCreate</code></td>
<td>2-3, 3-73</td>
</tr>
<tr>
<td><code>SC_PROCUSRINTCREATESTATIC</code></td>
<td>4-2</td>
</tr>
<tr>
<td><code>SC_PROCVARDEL</code></td>
<td>4-3</td>
</tr>
<tr>
<td><code>sc.procVarDel</code></td>
<td>2-6, 3-75</td>
</tr>
<tr>
<td><code>SC_PROCVARGET</code></td>
<td>4-3</td>
</tr>
<tr>
<td><code>sc.procVarGet</code></td>
<td>2-6, 3-76</td>
</tr>
<tr>
<td><code>SC_PROCVARINIT</code></td>
<td>4-2</td>
</tr>
<tr>
<td><code>sc.procVarInit</code></td>
<td>2-6, 3-77</td>
</tr>
<tr>
<td><code>SC_PROCVARRM</code></td>
<td>4-3</td>
</tr>
<tr>
<td><code>sc.procVarRm</code></td>
<td>2-6, 3-78</td>
</tr>
<tr>
<td><code>SC_PROCVARSET</code></td>
<td>4-3</td>
</tr>
<tr>
<td><code>sc.procVarSet</code></td>
<td>2-6, 3-79</td>
</tr>
<tr>
<td><code>sc.procVectorGet</code></td>
<td>2-3, 3-80</td>
</tr>
<tr>
<td><code>SC_PROCYIELD</code></td>
<td>4-2</td>
</tr>
<tr>
<td><code>sc.procYield</code></td>
<td>2-3, 3-81</td>
</tr>
<tr>
<td><code>SC_SET_MSGRX_HOOK</code></td>
<td>3-27</td>
</tr>
<tr>
<td><code>SC_SET_MSGTX_HOOK</code></td>
<td>3-27, 3-39</td>
</tr>
<tr>
<td><code>SC_SET_POOLCREATE_HOOK</code></td>
<td>3-40</td>
</tr>
<tr>
<td><code>SC_SET_POOLTX_HOOK</code></td>
<td>3-40</td>
</tr>
<tr>
<td><code>SC_SET_POCKILL_HOOK</code></td>
<td>3-50</td>
</tr>
<tr>
<td><code>SC_SET_PROCCREATE_HOOK</code></td>
<td>3-50</td>
</tr>
<tr>
<td><code>SC_SET_PROCSTOP_HOOK</code></td>
<td>3-50</td>
</tr>
<tr>
<td><code>SC_SET_PROCSWAP_HOOK</code></td>
<td>3-50</td>
</tr>
<tr>
<td><code>SC_SLEEP</code></td>
<td>4-3</td>
</tr>
<tr>
<td><code>sc.sleep</code></td>
<td>2-5</td>
</tr>
<tr>
<td><code>SC_SYSPOOLKILL</code></td>
<td>2-5</td>
</tr>
<tr>
<td><code>SC_SYSPOPCREATE</code></td>
<td>4-2</td>
</tr>
<tr>
<td><code>sc_tick</code></td>
<td>2-5, 3-83</td>
</tr>
<tr>
<td><code>sc_tickGet</code></td>
<td>2-5, 3-84</td>
</tr>
<tr>
<td><code>sc.tickLength</code></td>
<td>2-5, 3-85</td>
</tr>
<tr>
<td><code>sc.tickMs2Tick</code></td>
<td>2-5, 3-86</td>
</tr>
<tr>
<td><code>sc.tickTick2Ms</code></td>
<td>2-5, 3-87</td>
</tr>
<tr>
<td>SC_TMO .................................................................</td>
<td>4-3</td>
</tr>
<tr>
<td>SC_TMO_MAX ..........................................................</td>
<td>3-24</td>
</tr>
<tr>
<td>SC_TMOADD ...........................................................</td>
<td>4-3</td>
</tr>
<tr>
<td>sc_tmoAdd ...................................................................</td>
<td>2-5, 3-88</td>
</tr>
<tr>
<td>SC_TMORM ...............................................................</td>
<td>4-3</td>
</tr>
<tr>
<td>sc_tmoRm ...................................................................</td>
<td>2-5, 3-89</td>
</tr>
<tr>
<td>SC_TRIGGER ..............................................................</td>
<td>4-3</td>
</tr>
<tr>
<td>sc_trigger ...................................................................</td>
<td>2-5, 3-90</td>
</tr>
<tr>
<td>SC_TRIGGERVALUEGET ..................................................</td>
<td>4-3</td>
</tr>
<tr>
<td>sc_triggerValueGet ....................................................</td>
<td>2-5, 3-91</td>
</tr>
<tr>
<td>SC_TRIGGERVALUERESET ............................................</td>
<td>4-3</td>
</tr>
<tr>
<td>sc_triggerValueReset ..................................................</td>
<td>2-5, 3-92</td>
</tr>
<tr>
<td>SC_TRIGGERWAIT ..........................................................</td>
<td>4-3</td>
</tr>
<tr>
<td>sc_triggerWait ...........................................................</td>
<td>2-5, 3-93</td>
</tr>
<tr>
<td>Scheduler lock counter ..................................................</td>
<td>3-64, 3-65</td>
</tr>
<tr>
<td>Sender ........................................................................</td>
<td>3-32</td>
</tr>
<tr>
<td>SFS .................................................................</td>
<td>1-1</td>
</tr>
<tr>
<td>SFS_ATTR_DIR ..........................................................</td>
<td>3-1, 3-7, 3-19, 3-23, 3-59, 3-62, 3-66, 3-67</td>
</tr>
<tr>
<td>SMMS .................................................................</td>
<td>1-1</td>
</tr>
<tr>
<td>Snap-shot ....................................................................</td>
<td>3-42</td>
</tr>
<tr>
<td>Start a process ..........................................................</td>
<td>3-68</td>
</tr>
<tr>
<td>Start/stop counter .......................................................</td>
<td>3-68, 3-69</td>
</tr>
<tr>
<td>Stopped .......................................................................</td>
<td>3-68, 3-69</td>
</tr>
<tr>
<td>Supervise ....................................................................</td>
<td>3-57, 3-72</td>
</tr>
<tr>
<td>Supervisor process .......................................................</td>
<td>3-57</td>
</tr>
<tr>
<td>System call reference ..................................................</td>
<td>3-1</td>
</tr>
<tr>
<td>System calls overview ...................................................</td>
<td>2-1</td>
</tr>
<tr>
<td>System error ..................................................................</td>
<td>3-7</td>
</tr>
<tr>
<td>System tick calls ..........................................................</td>
<td>2-5</td>
</tr>
<tr>
<td>System tick length ........................................................</td>
<td>3-85</td>
</tr>
<tr>
<td>System ticks ..............................................................</td>
<td>3-82, 3-84, 3-86, 3-87</td>
</tr>
</tbody>
</table>

**T**

| Target manual ................................................................. | 1-1 |
| Technical specification .................................................. | 1-1 |
| Tick counter ..................................................................... | 3-84 |
| Time management ........................................................... | 1-2, 1-3 |
| Time-out expired ............................................................ | 3-89 |
| Timer process ............................................................... | 3-45, 3-66, 3-67, 3-70 |
| Timing calls .................................................................... | 2-5 |
| tmo ............................................................................ | 3-23, 3-30 |
| Trigger .......................................................................... | 3-90, 3-91, 3-92, 3-93 |

**U**

| Unlock ........................................................................... | 3-65 |
| Unused memory ............................................................. | 3-44 |
| User interrupt process .................................................... | 3-73 |

**W**

| Waiting ........................................................................... | 3-82 |
| Waiting state ............................................................... | 3-82 |
6 Index

Wanted ................................................................................................................................. 3-30

Y
Yield ................................................................................................................................. 3-81