Threads Extension for Portable Operating Systems: "Thin" Ada Binding

by

Ted Giering and Ted Baker
Department of Computer Science B-173
Florida State University
Tallahassee, FL 32306-4019

giering@cs.fsu.edu

for

U.S. Army HQ CECOM, Center for Software Engineering

15 December, 1992

DRAFT 0.1

The views contained in this report are those of the author. They should not be construed as an official position of the Department of the Army or the IEEE POSIX working group.

Acknowledgements

The preparation of this report was funded in part by the U.S. Army Communications Command (CECOM), initially through the U.S. Army Research Office and Battelle Laboratories, and later through Telos Corporation. It draws heavily on the work of the 1003.4 (Realtime Extension) and 1003.5 (Ada-language binding) groups of the IEEE POSIX standards organization.

Thanks are extended to the IEEE 1003.5 working group for reviewing drafts of this report.

Reprint Permissions

The U.S. Government is granted unlimited permission to reproduce this report, for Government purposes.

Permission is hereby granted for IEEE Standards Committee participants to reproduce this document for purposes of IEEE standardization activites. Permission is also granted for member bodies and technical committees of ISO an IEC to reproduce this document for purposes of developing a national position. Permission is granted to the IEEE to include all or portions of this work in standards documents produced by the IEEE. All other rights are reserved by the author. Use of information contained in this unapproved draft is at your own risk.

History of this Draft

Draft 0.0 (November 1992) This document started as a set of proposed new chapters for the P1003.20 binding. The ideas were presented at the POSIX meeting held in Utrecht, Netherlands, October 1992. Version 0.0 represents a first attempt to make a complete binding out of these chapters, to make it a separate document, and to incorporate corrections to the original chapters.

Draft 0.1 (December 1992) Version 0.1 changes the format to agree with the new format of P1003.20 Draft 1.1, which is planned for mock ballot. Several minor errors in parameter modes were also corrected, and an index was added.

Contents

PAGE

Contents i

Section 1: General 1
1.1 Introduction 1
1.2 Purpose 2
1.3 Scope 2
1.4 Sources 2
1.5 Disclaimer 2
1.6 Required Supporting Documents 2
1.7 Organization 3
1.8 Rationale for Ada Binding Approach 3
1.8.1 Threads and Tasks 3
1.8.2 Implementation-Defined Extensions 5
1.8.3 Extensions to P1003.5 5
1.8.4 Organizational Problems 6

Section 2: Terminology and General Requirements 7
2.1 Configurable Options and Limits 7
2.1.1 Configuration Options 7
2.1.2 Limits 9
2.2 Rationale 9
2.2.1 Task Specific Data 9

Section 4: Revisions to Process Environment 11
4.8 Configurable System Variables 11
4.8.1 Get Configurable System Variables 11
4.9 Rationale 12
4.9.1 Packaging 12

Section 17: Synchronization Primitives 13
17.1 Data Types 14
17.1.1 Mutex Attributes Type 14
17.1.2 Mutex Type 14
17.1.3 Condition Variable Attributes Type 14
17.1.4 Condition Variable Type 15

17.1.5 Mutex Protocol Type 15
17.2 Synchronization Primitives 16
17.3 Synchronization Overview 16
17.3.1 Blocking and Unblocking 16
17.3.2 Owner of a Mutex 16
17.4 Functions 17
17.4.1 Mutex Initialization Attributes 17
17.4.2 Initializing and Destroying a Mutex 17
17.4.3 Locking and Unlocking a Mutex 19
17.4.4 Condition Initialization Attributes 19
17.4.5 Initializing and Destroying Conditions 20
17.4.6 Broadcasting and Signaling a Condition 21
17.4.7 Waiting on a Condition 21
17.4.8 Changing Mutex Attributes 22
17.5 Rationale Relating to Synchronization Primitives 22
17.5.1 Chapter Organization 22
17.5.2 Intended Use of Synchronization Primitives 23
17.5.3 Mutexes, Condition Variables, and Scheduling 23
17.5.4 Ada 9X Considerations 24

Section 21: Task Priority Scheduling 27
21.1 Ada and POSIX Priorities 27
21.2 Data Types 27
21.2.1 Task Attributes Type 27
21.2.2 Contention Scope Type 28
21.3 Functions 28
21.3.1 Task Scheduling Attributes 28
21.3.2 Dynamic Task Scheduling Attributes Access 29
21.4 Rationale Relating to Priority Scheduling 31
21.4.1 Chapter Organization 31
21.4.2 Ada and POSIX Priority Scheduling 31

Section 26: Task Specific Data 33
26.1 Data Types 33
26.1.1 Task Identification Type 33
26.2 Functions 34
26.2.1 Get Calling Task's ID 34

Section A: Bibliography 35

Alphabetic Topical Index 37

Summary

This report describes a possible Ada-language interface to the proposed IEEE standard Threads Extension for Portable Operating Systems (P1003.4a). The POSIX Threads Extension is intended to promote the portability and reuse of applications using multiple threads of control within a process. The purpose of this report is to provide a point of departure for the development of such a standard Ada-language binding for P1003.4a, and to expose some of the technical issues involved.

Section 1: General

1.1 Introduction

This report describes a possible Ada-language interface to the proposed IEEE standard Threads Extension for Portable Operating Systems (POSIX) [10], which we hereafter refer to as P1003.4a or Pthreads. P1003.4a is a proposed extension to IEEE Std 1003.1-1990, hereafter referred to as P1003.1[7]. The Threads Extension provides services for the creation and management of multiple threads of control within a POSIX process. These threads of control share the address space, file descriptors, and other state information of the process in which they are created. They can also serve as an interface between a program and multiprocessor capabilities. By providing a standard interface to these services, it promotes portability and reuse of applications requiring a lighter weight method of multiprogramming than POSIX processes.

An important class of application requiring these services are real-time programs. For this reason, P1003.4a was developed by the POSIX Real-Time working group, which also developed the Real-Time Extensions for Portable Operating Systems (POSIX) [9], hereafter referred to as P1003.4. For this reason, P1003.4a derives some of its features from P1003.4, e.g. priority scheduling, and has features of its own of interest largely to the real-time community, e.g. priority inheritance.

The P1003.1 standard, on which P1003.4 and P1003.4a are based, exists presently only as a C-language binding, but working groups have been authorized to produce an Ada-language binding (hereafter referred to as P1003.5) and a language-independent specification (LIS). As of October 1992, the Ada-language binding has been approved as an IEEE standard, but has not yet appeared in print. The first balloting period for the P1003.1 LIS has just closed.

Like the base standard, the POSIX Real-Time Extension (P1003.4) and POSIX Threads Extension (P1003.4a) exists only as a C language interfaces, which are now going through the IEEE ballot process. Some work was started on an LIS for P1003.4, but this has not kept up with changes to the base document. The IEEE has authorized work on an Ada-language binding for P1003.4. This project authorization, which is designated P1003.20, also covers a binding for P1003.4a. However, P1003.20 has gone through 7 drafts without bindings to P1003.4a services, and will go to mock ballot on December 1. It has not been decided whether the binding to P1003.4a services should be a separate document or be included with the Real-Time Extensions binding document. For this reason, and because these bindings have not been reviewed by the POSIX.5 working group, this initial draft is being released as a separate document.

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

1.2 Purpose

The intended purpose of this document, which has been prepared as a technical report for the U.S. Army CECOM, is to provide a basis for evaluation of whether a standard Ada binding for the POSIX Threads Extension would be useful, to provide a starting point for the development of such a standard by the IEEE working group, and to expose some of the technical issues involved in the development of such a standard.

1.3 Scope

The scope of this report is limited to P1003.4a.

1.4 Sources

In addition to Draft 6 of P1003.4a [10], this report is based on IEEE Std 1003.1-1990 [7], Draft 13 of P1003.4, Draft 9 of P1003.5 [8], and Draft 0.6 of P1003.20 [4]. It is also based on the Ada language standard [11] and Draft 4.0 of the Ada 9X Mapping Specification [1].

1.5 Disclaimer

No one should specify or claim conformance to the interface described in this report. The views contained in this report are those of the author. They should not be construed as an official position of the Department of the Army or the IEEE POSIX organization. Moreover, the interface described in this report is subject to revision in future reports, due to changes in the views of the author and ongoing changes to the draft documents on which this report is based.

During work on this report there has been ample evidence of the magnitude of changes that can be expected to be required of Ada-language bindings, due to changes to the base documents and the policies of standards-making organizations.

The Ada language standard is also a moving target. The interface described here is based on the existing Ada language standard (Ada 83 [11]), but a revision process is now under way. The timetable calls for the new Ada language standard (Ada 9X) to be out around 1993. Given that it is unlikely that an Ada binding to P1003.4a is unlikely to be adopted by 1993, it seems that work on an Ada binding for P1003.4 should target Ada 9X. Fortunately, the differences between Ada 83 and Ada 9X are expected to be mostly upward-compatible, so large changes to the P1003.4a interface are not likely to be required.

1.6 Required Supporting Documents

This report is not self-contained. For the specification of the semantics of the Ada packages specified in this report, it is necessary to refer to Draft 6 of the IEEE

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

POSIX *Threads Extension for Portable Operating Systems*[9], and Draft 9 of the IEEE POSIX *Ada Language Interfaces Part I: Binding for System API*[8].

1.7 Organization

By not attempting to completely describe the semantics of the P1003.4a interface, this document exemplifies the so-called "thin" language binding approach. This is patterned after the P1003.20 binding to P1003.4, and is different from the precedent established by P1003.5, which is a "thick" binding – i.e. a completely self-contained description of the syntax and semantics of the P1003.1 Ada interface. The reasons for taking the thin binding approach here are discussed in § 1.8.5 of P1003.20 [4].

The organization of this report parallels that of the P1003.4a document. The present section consists of introductory material, including discussion of several global issues behind the design of the Ada-language binding. After that, the organization generally parallels that of P1003.4a. The order of presentation within subsections generally consists of the code of an Ada package specification, followed by text and tables explaining the correspondence of Ada names to P1003.4 C-Language Specification names. At the end of each major section, there is a subsection entitled "Rationale".

1.8 Rationale for Ada Binding Approach

The rationale subsections are provided to explain some of the reasoning behind the mapping from C to Ada, including uncertainties about interpretation of the P1003.4a interface, perceived defects in the present mapping, alternatives considered, and anticipated future interactions with the P1003.4a and Ada 9X standards activities.

The issues that pertain to the whole interface are discussed in the remainder of § 1.8. Issues pertaining to specific parts of the interface are then discussed within the sections that describe those parts.

1.8.1 Threads and Tasks

The original P1003.1 interface does not recognize the possibility of there being multiple concurrent threads of control within a process. The proposed P1003.4 Realtime Extension likewise does not recognize the existence of multiple threads within a process. However, the Ada language does allow for programs to be divided internally into multiple tasks. Tasks provide similar functionality to threads; they represent separate threads of control which share a common address space. The central question facing an Ada binding to P1003.4a is therefore what relationship tasks should have to threads.

In the absence of threads, Ada tasks must either be implemented as POSIX processes or interleaved within a process using a tasking kernel running within the process. It is also possible to do this in the presence of threads, that is, ignore threads and implement tasks directly. This would mean that both tasks and threads would have their own dispatcher, which would be multiplexing the same processor resources over different entities. It is difficult to see how threads and tasks could coexist in the

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

same process under these circumstances. This is a problem for Ada programs which use libraries which assume the presence of threads. It also limits the usefulness of an Ada binding to threads, since threads and tasks could not be intermixed.

Another possibility is to implement Ada tasks as threads, using Pthreads primitives in Ada runtime system (RTS) to create and control the execution of Ada tasks [6]. This avoids the use of two dispatchers, but there are still problems with binding to some Pthreads features [5]. Allowing Ada programs to create threads is a problem, because such threads, consisting as they do of Ada code, can perform tasking operations which normally require access to the calling task. For example, a thread could call an entry in a task. This typically requires the caller's Task Control Block to be placed on an entry queue, which presupposes the caller to be a task. While there are solutions to such problems, they can be costly to implement. There are also problems involved with the use of function pointers in the Pthreads interface, for which there is no portable Ada equivalent. These pointers are sometimes used for callbacks, where a function pointer is passed to the Pthreads interface, which later calls the function. Such callbacks to Ada code would have similar problems with tasking operations. Ada subprograms also support nested scopes, which presents the problem of providing them with appropriate environments when called from the Pthreads implementation.

Since tasks and threads provide similar services, an implementation of Ada tasks as threads provides much of the functionality of Pthreads. The difficulty of providing a safe, thin, direct binding to many Pthreads services has led to the conclusion that the Ada task facility should be the binding to Pthreads, and that additional bindings should be provided only to Pthreads capabilities which are useful extensions to Ada tasks and for which a safe, thin binding can be provided. All services in the binding are in terms of tasks; references to threads in the P1003.4a C-Language Specification are replaced by reference to tasks in the corresponding Ada binding.

This binding does not assume the presence of a Pthreads implementation in any other language; in particular, it does not assume that their is an implementation of the C binding specified in the P1003.4a C-Language Specification. An implementation of Ada tasking, together with the services described in this document, constitutes an implementation of POSIX threads. As such, this would leave open the possibility of more than one implementation of Pthreads on a single system, again making threads and tasks incompatible. Since it is the intent of this standard to allow for interoperability between languages using POSIX implementations, we specify that threads and tasks conforming to this standard must be able to be used together.

Such intermingling has been divided into three categories, depending on the degree to which tasks and threads can cooperate with each other:

1. **Coexistance** This is the ability of tasks and threads to execute within the same process. This might come about when an Ada program executes a C library function which creates a thread. The thread would be able to run, but the Ada program would not know of its existence or be able to manipulate it in any way.

2. **Visibility** Implies coexistance plus the ability of the Ada program to manipu-

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

late threads.

3. **Interoperability** This implies visibility, plus the ability of tasks and threads to fully interoperate; to be able to synchronize, share information, provide mutual exclusion, etc.

The exact definition of thread/task cooperation required by this standard is still being developed. They should at least coexist, and full interoperability is outside the scope of this standard. For the time being, this standard assumes that tasks and threads can coexist but cannot access one another in any way.

1.8.2 Implementation-Defined Extensions

P1003.5 has established precedents for the latitude an implementation is permitted in making changes to a package specification that is part of the standard. The only changes permitted to an implementation are:

a. adding **`with`** clauses, pragmas, representation specifications, and comments;

b. replacing "*implementation-defined*" by appropriate static value(s) or ranges;

c. adding or changing the packages' private parts;

d. any other changes that are lexically transparent to Ada compilers.

The intention behind these restrictions is to permit "with-clause portability". That is, it should be possible to determine whether an application relies on implementation-defined extensions by examining the with-clauses of the application. If no non-standard package names appear there, the application should be strictly conforming.

Of course, these restrictions cannot completely guarantee portability. The P1003.1, P1003.4, and P1003.4a standards do leave some semantic details unspecified. An application that depends on implementation semantics can be non-portable without using any non-standard names.

1.8.3 Extensions to P1003.5

The P1003.4a standard adds configuration options to the base standard. According to the IEEE TCOS-SSC draft Document Structure Plan of January 1992, there will be only one POSIX standard (LIS) document. Additions such as these will be merged into the appropriate sections of the base P1003.1 document.

If this principle is applied to P1003.5, it means adding to the specifications of the packages defined there. In this report, packages whose contents should be merged into a P1003.5 package are defined in a package of the same name. Whether to actually merge these packages is an issue that will need to be addressed by the P1003.5 working group.

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

1.8.4 Organizational Problems

The TCOS-SSC draft Document Structure Plan also calls for each language binding to parallel the structure of the LIS, down to the last paragraph. It is not clear whether it is practical to impose such a requirement on language bindings. The Ada-language binding for P1003.4a presented in this report reflects an attempt to parallel the structure of the P1003.4a document. This has not been entirely successful. The worst example is probably § 2, whose internal organization does not parallel that of P1003.4a at all. Part of the difficulty stems from the need for additional types and operations in the Ada interface, and part of it stems from the additional structure imposed by the Ada packages. In particular, there does not seem to be any place to put the Ada package specifications.

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

Section 2: Terminology and General Requirements

The package POSIX is extended to contain types used in testing for support of Pthreads features.

Specification of Package POSIX

```
package POSIX is
   -- Declarations carried forward from P1003.5 are not shown here.
     ...
   -- §2.1.1 Configuration Options   ...
end POSIX;
```

2.1 Configurable Options and Limits

2.1.1 Configuration Options

2.1.1.1 Synopsis

```
   subtype Threads_Support is Boolean range impl.-defined static range;
   subtype Stacksize_Support is Boolean range impl.-defined static range;
   subtype Priority_Inheritance_Support is Boolean range impl.-defined static range;
   subtype Priority_Ceiling_Support is Boolean range impl.-defined static range;
   subtype Process_Shared_Support is Boolean range impl.-defined static range;
```

2.1.1.2 Description

Certain features of this standard are optional. For each option, the P1003.4a C-Language Specification defines an optional compile-time constant, whose name is of the form _POSIX_*XXX*, and a configurable system variable, whose name is of the form _SC_POSIX_*XXX*.

The Ada-language interfaces corresponding to these options are divided into packages. In some cases, there are options within a package. The correspondence of the options to Ada-language packages is shown in Table 2.1.

The P1003.4a C-Language Specification specifies symbolic constants and configurable system variables that describe whether specific options are supported by the implementation. For each P1003.4a C-Language Specification symbolic constant:

If the range of the subtype is `True..True`, the option shall always be supported. If

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

Table 2.1 – Options & Packages

Option	Ada Package	Notes
{_POSIX_THREADS}	all threads packages	
{_POSIX_REENTRANT_FUNCTIONS}	*(none)*	(1)
{_POSIX_THREAD_ATTR_STACKSIZE}	*(none)*	(2)
{_POSIX_THREAD_PRIORITY_SCHEDULING}	`POSIX_Task_Attributes`	
{_POSIX_THREADS_PRIO_INHERIT}	`POSIX_Mutexes`	
{_POSIX_THREADS_PRIO_PROTECT}	`POSIX_Mutexes`	
{_POSIX_THREADS_PROCESS_SHARED}	`POSIX_Mutexes,POSIX_Condition_Variables`	

NOTES:

(1) Reentrancy is already provided by the 1003.5 versions of P1003.1 and P1003.4 non-reentrant functions.

(2) This option indicates whether the stack size attribute can be specified by an application. In this binding, it indicates whether the `storage_size` representation clause on tasks is implemented.

the range is `False..True` or `False..False` the function may not be supported[1]. If the range of the subtype is `False..True`, more information about whether the option is supported can be obtained by calling a function at run time (§4.8)[2].

The Ada-language subtypes for option support shall correspond to the P1003.4 C-Language Specification symbolic constants as shown in Table 2.2.

Table 2.2 – Compile-Time Ranges for Options

C-Language	Ada-Language
{_POSIX_THREADS}	`Threads_Support`
{_POSIX_THREAD_ATTR_STACKSIZE}	`Stacksize_Support`
{_POSIX_THREADS_PRIO_INHERIT}	`Priority_Inheritance_Support`
{_POSIX_THREADS_PRIO_PROTECT}	`Priority_Ceiling_Support`
{_POSIX_THREADS_PROCESS_SHARED}	`Process_Shared_Support`

The Ada-language functions to query whether options are supported are described in §4.8.1.

Note that there is no ada equivalent to {_POSIX_REENTRANT_FUNCTIONS}. This option specifies reentrant equivalents to non-reentrant functions defined in P1003.1. With multiple threads of control within a process, all OS operations must be reentrant, since they may be called by more than one thread, each of which expects the results of the operation to be unaffected by calls in other threads, except as specified in the standard. This is not true of a number of functions in P1003.1. For example, some of these functions return pointers to results which are written into static data areas in the process' address space. This reflects the limitation in some versions of the C language on the size of the object that can be returned as a function result.

[1] P1003.4a apparently allows the system to support an option, even when it replies `False` to queries whether the option is not supported.

[2] The function must always return `False` if the range is `False..False`, and always return `True` if the range is `True..True`.

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

Many of the limitations of the C language which lead to the definition of non-reentrant functions do not exist in Ada. In addition, P1003.5, the Ada binding to P1003.1, had to contend with multiple threads of control within a process from the beginning, so all P1003.5 operations are reentrant. For this reason, this binding to the P1003.4a C-Language Specification does not define reentrant equivalents to operations in P1003.5, and therefore does not need to define options indicating their availability.

2.1.1.3 References

Configurable System Variables, P1003.4a §2.6.3.

2.1.2 Limits

The only minimum imposed by the P1003.4a C-Language Specification is {DATAKEYS_MAX}. This is the portable minimum number of data keys which must be available for use with the thread specific data feature. This feature is not supported in this Ada binding, so this limit is not defined.

2.1.2.1 References

Configurable System Variables, P1003.4a §2.6.3; Thread Specific Data, P1003.4a §5.

2.2 Rationale

2.2.1 Task Specific Data

Task specific data is not included in the binding, since it is to some degree a duplication of task local data. It also requires a relatively thick binding to make safe, since the thread specific data is stored and retrieved with a single pair of functions as pointers untyped data. In addition to the problem of enforcing data types, there is the problem of representing destructor functions, which are function pointers associated with a key. There is no safe mechanism for representing subprogram pointers in Ada, so a thick binding is likely to be necessary.

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

98 *This side is intentionally blank.*

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

Section 4: Revisions to Process Environment

4.8 Configurable System Variables

Specification of Package POSIX_Configurable_System_Limits

The package POSIX_Configurable_System_Limits is extended to contain functions that may be used to obtain the values of limits imposed on the use of the interfaces specified by this standard that are imposed by the particular implementation.

```
with POSIX;
package POSIX_Configurable_System_Limits is
   -- Declarations carried forward from P1003.5 are not shown here.
     ...
   -- §4.8.1 Get Configurable System Variables   ...
end POSIX_Configurable_System_Limits;
```

The function to query whether an option is supported, whose name is of the form *XXX*_[Is/Are]_Supported, returns a value of the subtype POSIX_*XXX*_Support. If the function returns True, the option shall be supported. If the function returns False, whether the option is supported is unspecified.

4.8.1 Get Configurable System Variables

4.8.1.1 Synopsis

```
  function Threads_Are_Supported
     return POSIX.Threads_Support;
  function Stacksize_Is_Supported
     return POSIX.Stacksize_Support;
  function Priority_Inheritance_Is_Supported
     return POSIX.Priority_Inheritance_Support;
  function Priority_Ceiling_Is_Supported
     return POSIX.Priority_Ceiling_Support;
  function Process_Shared_Is_Supported
     return POSIX.Process_Shared_Support;
```

4.8.1.2 Description

The Ada-language functions for obtaining the values of configurable system variables shall correspond to configurable system variables of the P1003.4a C-Language Specification, as shown in Table 4.1.

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

Table 4.1 – Functions for Configurable System Variables

C-Language	Ada-Language
{_SC_THREADS}	`Threads_Are_Supported`
{_SC_THREAD_ATTR_STACKSIZE}	`Stacksize_Is_Supported`
{_SC_PRIO_INHERIT}	`Priority_Inheritance_Is_Supported`
{_SC_PRIO_PROTECT}	`Priority_Ceiling_Is_Supported`
{_SC_PROCESS_SHARED}	`Process_Shared_Is_Supported`

4.8.1.3 Error Handling

None of these functions shall raise any exception.

4.8.1.4 References

options, P1003.4 §2.9.3.

4.9 Rationale

4.9.1 Packaging

To follow the structure of the P1003.1 and P1003.4 documents, the declarations of the functions for configurable system have been inserted in the P1003.5 package `POSIX_Configurable_System_Limits`.

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

Section 17: Synchronization Primitives

This section describes the facilities relating to mutexes and condition variables available under the standard.

Specification of Package `POSIX_Mutexes`

```
with POSIX_Priority_Scheduling;
package POSIX_Mutexes is
   -- §17.1.1 Mutex Attributes Type ...
   -- §17.1.2 Mutex Type ...
   -- §17.1.5 Mutex Protocol Type ...
   -- §17.4.1 Mutex Initialization Attributes ...
   -- §17.4.8 Changing Mutex Attributes ...
   -- §17.4.2 Initializing and Destroying a Mutex ...
   -- §17.4.3 Locking and Unlocking a Mutex ...
private
   -- impl.-defined;
end POSIX_Mutexes;
```

Specification of Package `POSIX_Condition_Variables`

```
with POSIX_Mutexes;
with POSIX;
package POSIX_Condition_Variables is
   -- §17.1.3 Condition Variable Attributes Type ...
   -- §17.1.4 Condition Variable Type ...
   -- §17.4.4 Condition Initialization Attributes ...
   -- §17.4.5 Initializing and Destroying Conditions ...
   -- §17.4.6 Broadcasting and Signaling a Condition ...
   -- §17.4.7 Waiting on a Condition ...
private
   -- impl.-defined;
end POSIX_Condition_Variables;
```

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

17.1 Data Types

17.1.1 Mutex Attributes Type

17.1.1.1 Synopsis

```
type Attributes is private;
```

17.1.1.2 Description

The Ada-language name of the type for designating a mutex attributes object shall correspond to the P1003.4a C-Language Specification type as shown in Table 17.1.

Table 17.1 – Mutex Attributes Type

C-Language	Ada-Language
pthread_mutexattr_t	POSIX_Mutexes.Attributes

17.1.1.3 References

pthread_mutexattr_t, P1003.4a §2.4.

17.1.2 Mutex Type

17.1.2.1 Synopsis

```
type Mutex is private;
```

17.1.2.2 Description

The Ada-language name of the type for designating a mutex shall correspond to the P1003.4a C-Language Specification type as shown in Table 17.2.

Table 17.2 – Mutex Type

C-Language	Ada-Language
pthread_mutex_t	Mutex

17.1.2.3 References

pthread_mutex_t, P1003.4a §2.4.

17.1.3 Condition Variable Attributes Type

17.1.3.1 Synopsis

```
type Attributes is private;
```

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

17.1.3.2 Description

The Ada-language name of the type for designating a condition variable attributes object shall correspond to the P1003.4a C-Language Specification type as shown in Table 17.3.

Table 17.3 – Condition Variable Attributes Type

C-Language	Ada-Language
pthread_condattr_t	`POSIX_Condition_Variables.Attributes`

17.1.3.3 References

pthread_condattr_t, P1003.4a §2.4.

17.1.4 Condition Variable Type

17.1.4.1 Synopsis

```
type Condition is private;
```

17.1.4.2 Description

The Ada-language name of the type for designating a condition variable shall correspond to the P1003.4a C-Language Specification type as shown in Table 17.4.

Table 17.4 – Condition Variable Type

C-Language	Ada-Language
pthread_cond_t	`Condition`

17.1.4.3 References

pthread_mutex_t, P1003.4a §2.4.

17.1.5 Mutex Protocol Type

17.1.5.1 Synopsis

```
type Mutex_Protocol is impl.-defined static range;
No_Priority_Inheritance  : constant Mutex_Protocol := impl.-defined;
Highest_Blocked_Task     : constant Mutex_Protocol := impl.-defined;
Highest_Priority_Ceiling : constant Mutex_Protocol := impl.-defined;
```

17.1.5.2 Description

The Ada-language name of the type for designating the *protocol* mutex attribute shall correspond to the P1003.4a C-Language Specification type as shown in Table 17.5.

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

Table 17.5 – Mutex Protocol Type

C-Language	Ada-Language
pthread_protocol_t	`Mutex_Protocol`

The Ada-language names of the values defined in the P1003.4a C-Language Specification which the *protocol* mutex attribute can have shall correspond to the P1003.4a C-Language Specification values as shown in Table 17.6.

Table 17.6 – Mutex Protocol Type

C-Language	Ada-Language
NO_PRIO_INHERIT	`No_Priority_Inheritance`
PRIO_INHERIT	`Highest_Blocked_Task`
PRIO_PROTECT	`Highest_Priority_Ceiling`

17.1.5.3 References

protocol, P1003.4a §6.3.5.

17.2 Synchronization Primitives

17.3 Synchronization Overview

17.3.1 Blocking and Unblocking

Blocking is as defined in P1003.4a C-Language Specification, except that Ada tasks rather than threads are blocked.

17.3.2 Owner of a Mutex

A task becomes the owner of a mutex in the same way that a thread would, as described in P1003.4a C-Language Specification. However, the effect of aborting or terminating a task which holds a mutex are implementation defined, and must be documented for each implementation. The preferred semantics would be to protect users by automatically unlocking any mutexes owned by a terminating task. However, in the absence of such protection, it becomes the user's responsibility to ensure that all locks are unlocked before terminating.

Note that, in the absence of protection, a task that is aborted while waiting on a condition variable will own the mutex associated with the condition wait. This is implied by this section of the P1003.4a C-Language Specification, and explicitly stated in Section 9.3.3(180–183).

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

17.4 Functions

17.4.1 Mutex Initialization Attributes

17.4.1.1 Synopsis

```
  procedure Initialize(A : out Attributes);
  procedure Destroy(A : in out Attributes);
  function Get_Process_Shared(A : in Attributes) return Boolean;
  procedure Set_Process_Shared(A : in out Attributes; Process_Shared : in Boolean);
  procedure Set_Protocol (A : in out Attributes; Protocol : in Mutex_Protocol);
  function Get_Protocol (A : in Attributes) return Mutex_Protocol;
  procedure Set_Priority_Ceiling (A : in out Attributes;
       Priority_Ceiling : in POSIX_Priority_Scheduling.Scheduling_Priority);
  function Get_Priority_Ceiling (A : in Attributes)
      return POSIX_Priority_Scheduling.Scheduling_Priority;
```

17.4.1.2 Description

The Ada-language operations to initialize, destroy, and manipulate mutex attributes objects shall correspond to the P1003.4a C-Language Specification operations shown in Table 17.7.

17.4.1.3 References

pthread_mutexattr_init, *pthread_mutexattr_destroy*, *pthread_mutexattr_getpshared*, *pthread_mutexattr_setpshared*, P1003.4a § 4.4.1; *pthread_mutexattr_setprotocol*, *pthread_mutexattr_getprotocol*, *pthread_mutexattr_setprio_ceiling*, *pthread_mutexattr_getprio_ceiling*, P1003.4a § 6.3.5.

17.4.2 Initializing and Destroying a Mutex

17.4.2.1 Synopsis

```
  procedure Initialize(M : out Mutex; A : in Attributes);
  procedure Destroy(M : in out Mutex);
```

17.4.2.2 Description

The Ada-language operations to initialize and destroy a mutex shall correspond to the P1003.4a C-Language Specification operations shown in Table 17.8.

17.4.2.3 References

pthread_mutex_init, *pthread_mutex_destroy*, P1003.4a § 4.4.2.

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

Table 17.7 – Mutex Initialization Attributes

C-Language		Ada-Language		Notes
pthread_mutexattr_init		`Initialize`		
attr	*(O)*	`A`	*(O)*	
pthread_mutexattr_destroy		`Destroy`		
attr	*(I,O)*	`A`	*(I,O)*	
pthread_mutexattr_getpshared		`Get_Process_Shared`		
attr	*(I)*	`A`	*(I)*	
pshared	*(O)*	*<return-value>*	*(O)*	(1)
pthread_mutexattr_setpshared		`Set_Process_Shared`		
attr	*(I,O)*	`A`	*(I,O)*	
pshared	*(I)*	`Process_Shared`	*(I)*	(2)
pthread_mutexattr_setprotocol		`Set_Protocol`		
attr	*(I,O)*	`A`	*(I,O)*	
protocol	*(I)*	`Mutex_Protocol`	*(I)*	
pthread_mutexattr_getprotocol		`Get_Protocol`		
attr	*(I)*	`A`	*(I)*	
<return-value>		*<return-value>*		
pthread_mutexattr_setprio_ceiling		`Set_Priority_Ceiling`		
attr	*(I,O)*	`A`	*(I,O)*	
prio_ceiling	*(I)*	`Priority_Ceiling`	*(I)*	
pthread_mutexattr_getprio_ceiling		`Get_Priority_Ceiling`		
attr	*(I)*	`A`	*(I)*	
<return-value>		*<return-value>*		

NOTES:

(1) The Ada function returns `False` or `True` if the P1003.4a C-Language Specification function would return with *pshared* = 0 or *pshared* = 1, respectively.

(2) Passing the Ada procedure a `Process_Shared` parameter of `False` or `True` has the effect of passing the P1003.4a C-Language Specification function a *pshared* parameter of 0 or 1, respectively.

Table 17.8 – Initializing and Destroying a Mutex

C-Language		Ada-Language	
pthread_mutex_init		`Initialize`	
mutex	*(O)*	`M`	*(O)*
attr	*(I)*	`A`	*(I)*
pthread_mutex_destroy		`Destroy`	
mutex	*(I,O)*	`M`	*(I,O)*

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

17.4.3 Locking and Unlocking a Mutex

17.4.3.1 Synopsis

```
procedure Lock(M : in out Mutex);
procedure Try_Lock(M : in out Mutex; Success : out Boolean);
procedure Unlock(M : in out Mutex);
```

17.4.3.2 Description

The Ada-language operations to lock and unlock a mutex shall correspond to the P1003.4a C-Language Specification operations shown in Table 17.9.

Table 17.9 – Locking and Unlocking a Mutex

C-Language		Ada-Language		Notes
pthread_mutex_lock		`Lock`		
mutex	*(I,O)*	`M`	*(I,O)*	
pthread_mutex_trylock		`Try_Lock`		
mutex	*(I,O)*	`M`	*(I,O)*	
<return-value>	*(O)*	`Success`	*(O)*	(1)
pthread_mutex_unlock		`Unlock`		
mutex	*(I,O)*	`M`	*(I,O)*	

NOTES:

(1) The Ada procedure returns with `Success = False` if and only if the P1003.4a C-Language Specification operation would return a value of -1 with error code [EAGAIN].

17.4.3.3 Error Handling

The error handling is as specified in the P1003.4a C-Language Specification, except that no exception is raised for the P1003.4a C-Language Specification "error" code [EAGAIN], since in this case the function `Try_Lock` returns with `Success = False`.

17.4.3.4 References

pthread_mutex_lock, *pthread_mutex_trylock*, *pthread_mutex_unlock*, P1003.4a §4.4.2.

17.4.4 Condition Initialization Attributes

17.4.4.1 Synopsis

```
procedure Initialize(A : out Attributes);
procedure Destroy(A : in out Attributes);
function Get_Process_Shared(A : in Attributes) return Boolean;
procedure Set_Process_Shared(A : in out Attributes; Process_Shared : in Boolean);
```

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

17.4.4.2 Description

The Ada-language operations to initialize, destroy, and manipulate condition attributes objects shall correspond to the P1003.4a C-Language Specification operations shown in Table 17.10.

Table 17.10 – Condition Initialization Attributes

C-Language		Ada-Language		Notes
pthread_condattr_init		`Initialize`		
attr	*(O)*	`A`	*(O)*	
pthread_condattr_destroy		`Destroy`		
attr	*(I,O)*	`A`	*(I,O)*	
pthread_condattr_getpshared		`Get_Process_Shared`		
attr	*(I)*	`A`	*(I)*	
pshared	*(O)*	*<return-value>*	*(O)*	(1)
pthread_condattr_setpshared		`Set_Process_Shared`		
attr	*(I,O)*	`A`	*(I,O)*	
pshared	*(I)*	`Process_Shared`	*(I)*	(2)

NOTES:

(1) The Ada function returns `False` or `True` if the P1003.4a C-Language Specification function would return with *pshared* = 0 or *pshared* = 1, respectively.

(2) Passing the Ada procedure a `Process_Shared` parameter of `False` or `True` has the effect of passing the P1003.4a C-Language Specification function a *pshared* parameter of 0 or 1, respectively.

17.4.4.3 References

pthread_condattr_init, *pthread_condattr_destroy*, *pthread_condattr_getpshared*, *pthread_condattr_setpshared*, P1003.4a § 4.4.4.

17.4.5 Initializing and Destroying Conditions

17.4.5.1 Synopsis

```
procedure Initialize(C : out Condition; A : in Attributes);
procedure Destroy(C : in out Condition);
```

17.4.5.2 Description

The Ada-language operations to initialize and destroy a condition variable shall correspond to the P1003.4a C-Language Specification operations shown in Table 17.11.

17.4.5.3 References

pthread_cond_init, *pthread_cond_destroy*, P1003.4a § 4.4.5.

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

Table 17.11 – Initializing and Destroying Conditions

C-Language		Ada-Language	
pthread_cond_init		Initialize	
cond	*(O)*	C	*(O)*
attr	*(I)*	A	*(I)*
pthread_mutex_destroy		Destroy	
cond	*(I,O)*	C	*(I,O)*

17.4.6 Broadcasting and Signaling a Condition

17.4.6.1 Synopsis

```
procedure Signal(C : in out Condition);
procedure Broadcast(C : in out Condition);
```

17.4.6.2 Description

The Ada-language operations to unblock tasks waiting on condition variables shall correspond to the P1003.4a C-Language Specification operations shown in Table 17.12.

Table 17.12 – Broadcasting and Signaling a Condition

C-Language		Ada-Language	
pthread_cond_signal		Signal	
cond	*(I,O)*	C	*(I,O)*
pthread_cond_broadcast		Broadcast	
cond	*(I,O)*	C	*(I,O)*

17.4.6.3 References

pthread_cond_signal, *pthread_cond_broadcast*, P1003.4a § 4.4.6.

17.4.7 Waiting on a Condition

17.4.7.1 Synopsis

```
procedure Wait(C : in out Condition; M : in out POSIX_Mutexes.Mutex);
procedure Timed_Wait(C : in out Condition; M : in out POSIX_Mutexes.Mutex;
     Wakeup_At : in POSIX.Timespec);
```

17.4.7.2 Description

The Ada-language operations to wait on a condition variable shall correspond to the P1003.4a C-Language Specification operations shown in Table 17.13.

17.4.7.3 References

pthread_cond_wait, *pthread_cond_timedwait*, P1003.4a § 4.4.7.

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

Table 17.13 – Waiting on a Condition

C-Language		Ada-Language	
pthread_cond_wait		Wait	
cond	*(I,O)*	C	*(I,O)*
mutex	*(I,O)*	M	*(I,O)*
pthread_cond_timedwait		Timed_Wait	
cond	*(I,O)*	C	*(I,O)*
mutex	*(I,O)*	M	*(I,O)*
abstime	*(I)*	Wakeup_At	*(I)*

17.4.8 Changing Mutex Attributes

17.4.8.1 Synopsis

```
procedure Set_Priority_Ceiling (M : in Mutex;
     Priority_Ceiling : in POSIX_Priority_Scheduling.Scheduling_Priority);
function Get_Priority_Ceiling (M : in Mutex)
    return POSIX_Priority_Scheduling.Scheduling_Priority;
```

17.4.8.2 Description

The Ada-language operations to change the values of the attributes of existing mutexes shall correspond to the P1003.4a C-Language Specification operations shown in Table 17.14.

Table 17.14 – Changing Mutex Attributes

C-Language		Ada-Language	
pthread_mutex_getprio_ceiling		Get_Priority_Ceiling	
mutex	*(I)*	M	*(I)*
<return-value>		*<return-value>*	
pthread_mutex_setprio_ceiling		Set_Priority_Ceiling	
mutex	*(I,O)*	M	*(I,O)*
prio_ceiling	*(I)*	Priority_Ceiling	*(I)*

17.4.8.3 References

pthread_mutex_getprio_ceiling, *pthread_mutex_setprio_ceiling*, P1003.4a § 6.3.6

17.5 Rationale Relating to Synchronization Primitives

17.5.1 Chapter Organization

The operations relating to mutex scheduling protocols in § 17.4.1.2 are documented in Chapter 6 on Priority Scheduling in the P1003.4a C-Language Specification draft. However, since the related operations are on mutexes and mutex attribute objects, it is best to put these operations in the package that declares these objects for purposes of modularity. Since the mutex is documented in this chapter, it seems preferable to

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

describe these operations in this chapter, rather than refer back to this package in the Priority Scheduling chapter.

17.5.2 Intended Use of Synchronization Primitives

An interface to the Pthreads synchronization primitives is included in the binding based on the assumption that, for many applications on many implementations, they will be more efficient than the corresponding rendezvous primitives of Ada. This will almost certainly be true in the case of an Ada RTS built using Pthreads operations. In this case, a rendezvous will involve a wait on a condition variable as well as additional code to enforce rendezvous semantics. Such an implementation is outlined in Section 9.5.1 of the Pthreads draft [10] and in more detail in a note to the Pthreads draft [3] and in a description of such an implementation by Giering and Baker [6]. Unless an application actually needs the semantics of rendezvous, such that it would have to be implemented in any case, Pthreads synchronization primitives would always be more efficient in such an implementation.

It is possible to comply with this standard without implementing Ada tasking over an existing Pthreads implementation. Much of the semantics of Pthreads is subsumed by Ada tasking, and the synchronization functions specified here may be implemented by the Ada runtime rather than the other way around. In this case, it is less clear that the Pthreads synchronization primitives would have less overhead than the corresponding rendezvous operations. Much of the complexity of rendezvous results from interaction with other portions of the RTS, particularly abortion. This standard leaves protection of mutexes against abortion implementation defined. If such protection is provided, the use of mutexes and condition variables will be more expensive.

17.5.3 Mutexes, Condition Variables, and Scheduling

Ada allows the specification of task priorities using the `priority` pragma. ARM 9.8(4) gives the following rule:

> If two tasks with different priorities are both eligible for execution and could sensibly be executed using the same physical processors and the same other processing resources, then it cannot be the case that the task with the lower priority is executing while the task with the higher priority is not.

The use of condition variables and mutexes with the `No_Priority_Inheritance` policy (the default) suspends the task. This just expands the cases under which a task is not eligible for execution, and can be interpreted as complying with this rule. The use of mutexes with either of the `Highest_Blocked_Task` or `Highest_Priority_Ceiling` protocols, however, can modify a task's priority under circumstances other than those specified in the ARM. This can raise the priority of a task which owns a mutex, and can result in a task with a low priority (as assigned by the `priority` pragma) running at the expense of a higher priority task. This is not allowed by the

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

ARM. Use of mutexes with these scheduling policies must therefore be considered an extension to the language, and may result in illegal behavior.

17.5.4 Ada 9X Considerations

The current draft of the Ada 9X specification contains the *protected type*, a monitor-like construct designed to provide, among other things, lower-cost synchronization [2]. The protected type may obviate the need for mutexes and condition variables. Similar functionality to the mutex is provided by the protected procedure. This is a procedure within a protected type which can only be executed by one task at a time; other tasks attempting to call the procedure are suspended until its execution is complete. This allows for the protection of critical sections, much as a mutex does, and could be implemented by having the procedure lock a mutex on entry and unlock it on exit. An extra procedure call is involved, but this could be eliminated by having the compiler expand the critical section inline. The only functionality of mutexes not supported is the ability of a task to unlock mutexes in an arbitrary order; protected procedures must be nested. However, unlocking mutexes in other than LIFO order is considered by many to be a dubious programming practice.

Protected record entries provide similar functionality to that of condition variables. In the case of entries, the actual conditions that should be checked on returning from a condition wait are included in the language. For example, a condition wait such as

```
pthread_mutex_lock(mutex);
while(not condition) pthread_cond_wait(condition, mutex);
```

has a similar function to the Ada 9X construct

```
protected type Condition_Wait is
  entry Wait;
  procedure Signal;
end Condition_Wait;
protected body Condition_Wait is
  condition : BOOLEAN := False;
  entry Wait when condition is
  begin
    condition := False;
  end Wait;
  procedure Signal is
  begin
    condition := True;
  end Signal;
end Condition_Wait;
```

A task making a call on an entry whose guard is `False` are suspended until the guard becomes `True`. If there are tasks waiting on such an entry, exiting a protected procedure in the same protected type will automatically reevaluate the effected guards. If any become true, one waiting task is made eligible to execute. The Signal in this case makes the condition true and wakes the task up in a single operation, as opposed to Pthreads, which separates the signal from the condition being awaited.

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

This does mean that Ada 9X has no equivalent to a condition variable wait without an associated condition. However, since Pthreads provides no guarantee on the number of tasks awakened by a *pthread_cond_signal* operation, this is also a dubious programming practice.

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

404 *This side is intentionally blank.*

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

Section 21: Task Priority Scheduling

This section describes the facilities allowing the dynamic adjustment of task scheduling attributes under the standard.

Specification of Package `POSIX_Task_Attributes`

```
with POSIX_Priority_Scheduling;
with POSIX_Task_Identification;
with POSIX_Mutexes;
package POSIX_Task_Attributes is
   -- §21.2.1 Task Attributes Type   ...
   -- §21.2.2 Contention Scope Type   ...
   -- §21.3.1 Task Scheduling Attributes   ...
   -- §21.3.2 Dynamic Task Scheduling Attributes Access   ...

private
   -- impl.-defined;
end POSIX_Task_Attributes;
```

21.1 Ada and POSIX Priorities

Ada and POSIX define priority scheduling for tasks and threads, respectively. The Ada priority type is defined as an implementation defined subrange of type `Integer` (ARM 13.7(3)). The POSIX priority type is defined by this standard to be another implementation defined subrange of type `Integer`. The relationship between these two types is implementation defined.

21.2 Data Types

21.2.1 Task Attributes Type

21.2.1.1 Synopsis

```
type Task_Attributes is private;
```

21.2.1.2 Description

The Ada-language name of the type for designating a task attributes object shall correspond to the P1003.4a C-Language Specification type designating thread attributes as shown in Table 21.1.

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

Table 21.1 – Task Attributes Type

C-Language	Ada-Language
pthread_attr_t	Task_Attributes

21.2.1.3 References

pthread_attr_t, P1003.4a §2.4.

21.2.2 Contention Scope Type

21.2.2.1 Synopsis

```
type Contention_Scope is (Local_Scope, Global_Scope);
```

21.2.2.2 Description

The Ada-language name of the type for designating the *contentionscope* attribute shall correspond to the LIS type, and the the Ada-language names of the values defined in the P1003.4a C-Language Specification which the contentionscope attribute can have shall correspond to the P1003.4a C-Language Specification values as shown in Table 21.2.

Table 21.2 – Contention Scope Type

C-Language	Ada-Language	Notes
int	Contention_Scope	
PTHREAD_SCOPE_LOCAL	Local_Scope	
PTHREAD_SCOPE_GLOBAL	Global_Scope	

21.2.2.3 References

pthread_attr_setscope, *pthread_attr_getscope*, P1003.4a §6.3.1.

21.3 Functions

21.3.1 Task Scheduling Attributes

21.3.1.1 Synopsis

```
procedure Set_Contention_Scope
   (Attributes : in out Task_Attributes;
    New_Scope  : in Contention_Scope;
    Old_Scope  : out Contention_Scope);
function Get_Contention_Scope
   (Attributes : in Task_Attributes)
   return Contention_Scope;
procedure Set_Inherit_Scheduling
   (Attributes      : in out Task_Attributes;
```

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

```
63          New_Inheritance : in Boolean;
64          Old_Inheritance : out Boolean);
65       function Get_Inherit_Scheduling
66          (Attributes : in Task_Attributes)
67          return Boolean;
68       procedure Set_Policy
69          (Attributes : in out Task_Attributes;
70           Old_Policy : in POSIX_Priority_Scheduling.Scheduling_Policy;
71           New_Policy : out POSIX_Priority_Scheduling.Scheduling_Policy);
72       function Get_Policy
73          (Attributes : in Task_Attributes)
74          return POSIX_Priority_Scheduling.Scheduling_Policy;
75       procedure Set_Priority
76          (Attributes   : in out Task_Attributes;
77           New_Priority : in POSIX_Priority_Scheduling.Scheduling_Priority;
78           Old_Priority : out POSIX_Priority_Scheduling.Scheduling_Priority);
79       function Get_Priority
80          (Attributes : in Task_Attributes)
81          return POSIX_Priority_Scheduling.Scheduling_Priority;
```

82 21.3.1.2 Description

83 The Ada-language operations to access and modify scheduling attributes in a task at-
84 tributes object shall correspond to the P1003.4a C-Language Specification operations
85 shown in Table 21.3.

86 Note that the C-language error reporting mechanism implies that the Ada operations
87 can never return a value of -1, since this would raise an exception in the Aad interface.

88 21.3.1.3 References

89 *pthread_attr_setscope*, *pthread_attr_getscope*, *pthread_attr_setinheritsched*, *pthread_-*
90 *attr_getinheritsched*, *pthread_attr_setsched*, *pthread_attr_getsched*, *pthread_attr_-*
91 *setprio*, *pthread_attr_getprio*, P1003.4a § 6.3.1.

92 21.3.2 Dynamic Task Scheduling Attributes Access

93 21.3.2.1 Synopsis

```
94       function Get_Attributes
95          (T : POSIX_Task_Identification.Task_ID)
96          return Task_Attributes;
97       procedure Set_Attributes
98          (T : in out POSIX_Task_Identification.Task_ID;
99           Attributes : in Task_Attributes);
```

100 21.3.2.2 Description

101 The Ada-language operations to retrieve and set the scheduling attributes of a task
102 shall correspond to the P1003.4a C-Language Specification operations to retrieve
103 and set the scheduling attributes of a thread as shown in Table 21.4.

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

Table 21.3 – Task Scheduling Attributes

C-Language		Ada-Language		Notes
pthread_attr_setscope		`Set_Contention_Scope`		
attr	*(I,O)*	`Attributes`	*(I,O)*	
attr	*(I)*	`New_Scope`	*(I)*	
<return-value>	*(O)*	`Old_Scope`	*(O)*	
pthread_attr_setscope		`Set_Contention_Scope`		
contentionscope	*(I)*	`Scope`	*(I)*	
pthread_attr_getscope		`Get_Contention_Scope`		
attr	*(I)*	`Attributes`	*(I)*	
<return-value>		*<return-value>*		
pthread_attr_setinheritsched		`Set_Inherit_Scheduling`		
attr	*(I,O)*	`Attributes`	*(I,O)*	
inherit	*(I)*	`New_Inheritance`	*(I)*	(1)
<return-value>	*(O)*	`Old_Inheritance`	*(O)*	(2)
pthread_attr_getinheritsched		`Get_Inherit_Scheduling`		
attr	*(I)*	`Attributes`	*(I)*	
inherit	*(O)*	*<return-value>*	*(O)*	(2)
pthread_attr_setsched		`Set_Policy`		
attr	*(I,O)*	`Attributes`	*(I,O)*	
scheduler	*(I)*	`New_Policy`	*(I)*	
<return-value>	*(O)*	`Old_Policy`	*(O)*	
pthread_attr_getsched		`Get_Policy`		
attr	*(I)*	`Attributes`	*(I)*	
<return-value>		*<return-value>*		
pthread_attr_setprio		`Set_Priority`		
attr	*(I,O)*	`Attributes`	*(I,O)*	
priority	*(I)*	`New_Priority`	*(I)*	
<return-value>	*(O)*	`Old_Priority`	*(O)*	
pthread_attr_getprio		`Get_Priority`		
attr	*(I)*	`Attributes`	*(I)*	
<return-value>		*<return-value>*		

NOTES:

(1) Passing the Ada procedure a `Inherit_Scheduling` parameter of `False` or `True` has the effect of passing the P1003.4a C-Language Specification function a *pshared* parameter of PTHREAD_INHERIT_SCHED or PTHREAD_DEFAULT_SCHED, respectively.

(2) The Ada operations return `False` or `True` if the P1003.4a C-Language Specification function would return the value PTHREAD_INHERIT_SCHED or PTHREAD_DEFAULT_SCHED, respectively.

Table 21.4 – Dynamic Task Scheduling Attributes Access

C-Language		Ada-Language	
pthread_getschedattr		`Get_Attributes`	
thread	*(I)*	`T`	*(I)*
attrs	*(O)*	*<return-value>*	*(O)*
pthread_setschedattr		`Set_Attributes`	
thread	*(I)*	`T`	*(I)*
attrs	*(I,O)*	`Attributes`	*(I,O)*

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

21.3.2.3 References

pthread_getschedattr, *pthread_setschedattr*, P1003.4a § 6.3.2.

21.4 Rationale Relating to Priority Scheduling

21.4.1 Chapter Organization

Operations on mutex scheduling protocols is documented in this chapter of the P1003.4a C-Language Specification. In this binding, these operations have been moved to Chapter 17, as mentioned in the rational for that chapter.

21.4.2 Ada and POSIX Priority Scheduling

Ada allows the specification of task priorities using the `priority` pragma. ARM 9.8(6) states that these priorities are static, that is, they cannot be explicitly changed during runtime. The priority of a task only changes when it accepts an entry call from a higher priority task, in which case it will execute the entry body at the higher priority. The use of Pthreads primitives to dynamically adjust the priority of a task may therefore violate Ada tasking semantics.

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

165 *This side is intentionally blank.*

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

Section 26: Task Specific Data

This section describes the facilities allowing the identification of tasks to operations defined under the standard.

Specification of Package `POSIX_Task_Identification`

```
package POSIX_Task_Identification is
   -- §26.1.1 Task Identification Type  ...
   -- §26.2.1 Get Calling Task's ID  ...
private
   -- impl.-defined;
end POSIX_Task_Identification;
```

26.1 Data Types

26.1.1 Task Identification Type

26.1.1.1 Synopsis

```
type Task_ID is private;
```

26.1.1.2 Description

The Ada-language name of the type for designating a task object shall correspond to the P1003.4a C-Language Specification type designating a thread object as shown in Table 26.1.

Table 26.1 – Task Identification Type

C-Language	Ada-Language
pthread_t	`Task_ID`

26.1.1.3 References

pthread_t, P1003.4a §2.4.

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

26.2 Functions

26.2.1 Get Calling Task's ID

26.2.1.1 Synopsis

```
function Current_Task return Task_ID;
```

26.2.1.2 Description

The Ada-language operation to access the task identifier of the calling task shall correspond to the P1003.4a C-Language Specification operation to access the thread ID of the calling thread as shown in Table 26.2.

Table 26.2 – Retrieving and Setting Scheduling Attributes

C-Language	Ada-Language
pthread_self <*return-value*>	`Current_Task` <*return-value*>

26.2.1.3 References

pthread_self, P1003.4a §3.3.6.

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

Section A
(informative)
Bibliography

The following documents are related to this proposal in an informative manner, or were used as references in the preparation of this proposal.

{1} Ada 9X Project Office. *Ada 9X Mapping Document, Volume II – Mapping Specification*, December 1991.

{2} Ada 9X Project Office. *Ada 9X Integrated Language Specification*, September 1992.

{3} Ted Baker, Bob Conti, Offer Pazy, and Harold Seigel. An approach to implementing Ada rendezvous using Pthreads. POSIX note P1003.4-N0261, October 1990.

{4} T.P. Baker. Realtime extension for portable operating systems: "thin" Ada binding (draft 1). Technical report, Florida State University, Department of Computer Science, Tallahassee, Florida, 1992.

{5} E.W. Giering and T.P. Baker. POSIX/Ada real-time bindings: Description of work in progress. In *Proceedings of the Ninth Annual Washington Ada Symposium*. ACM, July 1992.

{6} E.W. Giering and T.P. Baker. Using POSIX threads to implement Ada tasking: Description of work in progress. In *TRI-Ada '92 Proceedings*. ACM, 1992. To appear.

{7} IEEE. *Information Technology—Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API) [C Language]*, 1990. IEEE Std 1003.1-1990.

{8} IEEE. *Portable Operating System Interface for Computer Environments, Ada Binding (Draft 8)*, March 1992. P1003.5/D8.

{9} IEEE. *Realtime Extension for Portable Operating Systems (Draft 12)*, February 1992. P1003.4/D12.

{10} IEEE. *Threads Extension for Portable Operating Systems (Draft 6)*, February 1992. P1003.4a/D6.

{11} United States Department of Defense, Ada Joint Program Office. *Reference Manual for the Ada Programming Language*, February 1983. ANSI/MIL-STD-1815A-1983.

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

This side is intentionally blank.

Alphabetic Topical Index

A

Attributes
type ... 14–15, 37

B

Broadcast
procedure ... 21, 37

C

Condition
type ... 15, 37
Contention_Scope
type ... 28, 37
contentionscope
... 28, 37
Current_Task
function ... 34, 37

D

{DATAKEYS_MAX}
limit ... 9, 37
Destroy
procedure ... 18, 20–21, 37

E

[EAGAIN]
error status value ... 19, 37

F

G

Get_Attributes
function ... 30, 37
Get_Contention_Scope
function ... 30, 37
Get_Inherit_Scheduling
function ... 30, 37
Get_Policy
function ... 30, 37
Get_Priority
function ... 30, 37
Get_Priority_Ceiling
function ... 18, 22, 37
Get_Process_Shared
function ... 18, 20, 37
Get_Protocol
function ... 18, 37
Global_Scope
enumeration literal ... 28, 37

H

Highest_Blocked_Task
constant ... 16, 23, 37
Highest_Priority_Ceiling
constant ... 16, 23, 37

I

Initialize
procedure ... 18, 20–21, 37
int
type ... 28, 37

J

K

L

Local_Scope
enumeration literal ... 28, 37
Lock
procedure ... 19, 37

M

Mutex
type ... 14, 37
Mutex_Protocol
type ... 16, 37

N

NO_PRIO_INHERIT
constant ... 16, 37
No_Priority_Inheritance
constant ... 16, 23, 37

O

P

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

POSIX
package ... 7, 38
POSIX_Condition_Variables
package ... 8, 13, 38
POSIX_Configurable_System_Limits
package ... 11–12, 38
POSIX_Mutexes
package ... 8, 13, 38
{_POSIX_REENTRANT_FUNCTIONS}
limit ... 8, 38
POSIX_Task_Attributes
package ... 8, 27, 38
POSIX_Task_Identification
package ... 33, 38
{_POSIX_THREAD_ATTR_STACKSIZE}
limit ... 8, 38
{_POSIX_THREAD_PRIORITY_SCHEDULING}
limit ... 8, 38
{_POSIX_THREADS}
limit ... 8, 38
{_POSIX_THREADS_PRIO_INHERIT}
limit ... 8, 38
{_POSIX_THREADS_PRIO_PROTECT}
limit ... 8, 38
{_POSIX_THREADS_PROCESS_SHARED}
limit ... 8, 38
PRIO_INHERIT
constant ... 16, 38
PRIO_PROTECT
constant ... 16, 38
priority
pragma ... 23, 31, 38
Priority_Ceiling_Is_Supported
function ... 12, 38
Priority_Ceiling_Support
subtype ... 8, 38
Priority_Inheritance_Is_Supported
function ... 12, 38
Priority_Inheritance_Support
subtype ... 8, 38
Process_Shared_Is_Supported
function ... 12, 38
Process_Shared_Support
subtype ... 8, 38
pthread_attr_getinheritsched
function ... 29–30, 38
pthread_attr_getprio
function ... 29–30, 38
pthread_attr_getsched
function ... 29–30, 38
pthread_attr_getscope
function ... 28–30, 38
pthread_attr_setinheritsched
function ... 29–30, 38
pthread_attr_setprio
function ... 29–30, 38
pthread_attr_setsched
function ... 29–30, 38
pthread_attr_setscope
function ... 28–30, 38
pthread_attr_t
type ... 28, 38
pthread_condattr_destroy
function ... 20, 38
pthread_condattr_getpshared
function ... 20, 38
pthread_condattr_init
function ... 20, 38
pthread_condattr_setpshared
function ... 20, 38
pthread_condattr_t
type ... 15, 38
pthread_cond_broadcast
function ... 21, 38
pthread_cond_destroy
function ... 20, 38
pthread_cond_init
function ... 20–21, 38
pthread_cond_signal
function ... 21, 25, 38
pthread_cond_t
type ... 15, 38
pthread_cond_timedwait
function ... 21–22, 38
pthread_cond_wait
function ... 21–22, 38
PTHREAD_DEFAULT_SCHED
constant ... 30, 38
pthread_getschedattr
function ... 30–31, 38
PTHREAD_INHERIT_SCHED
constant ... 30, 38
pthread_mutexattr_destroy
function ... 17–18, 38
pthread_mutexattr_getprio_ceiling
function ... 17–18, 38
pthread_mutexattr_getprotocol
function ... 17–18, 38
pthread_mutexattr_getpshared
function ... 17–18, 38
pthread_mutexattr_init
function ... 17–18, 38
pthread_mutexattr_setprio_ceiling
function ... 17–18, 38
pthread_mutexattr_setprotocol
function ... 17–18, 38
pthread_mutexattr_setpshared
function ... 17–18, 38
pthread_mutexattr_t
type ... 14, 38
pthread_mutex_destroy
function ... 17–18, 21, 38
pthread_mutex_getprio_ceiling
function ... 22, 38
pthread_mutex_init

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.

function ... 17–18, 38
pthread_mutex_lock
function ... 19, 39
pthread_mutex_setprio_ceiling
function ... 22, 39
pthread_mutex_t
type ... 14–15, 39
pthread_mutex_trylock
function ... 19, 39
pthread_mutex_unlock
function ... 19, 39
pthread_protocol_t
type ... 16, 39
PTHREAD_SCOPE_GLOBAL
constant ... 28, 39
PTHREAD_SCOPE_LOCAL
constant ... 28, 39
pthread_self
function ... 34, 39
pthread_self
type ... 34, 39
pthread_setschedattr
function ... 30–31, 39
pthread_t
type ... 33, 39

Q

R

S

{_SC_PRIO_INHERIT}
limit ... 12, 39
{_SC_PRIO_PROTECT}
limit ... 12, 39
{_SC_PROCESS_SHARED}
limit ... 12, 39
{_SC_THREAD_ATTR_STACKSIZE}
limit ... 12, 39
{_SC_THREADS}
limit ... 12, 39
Set_Attributes
procedure ... 30, 39
Set_Contention_Scope
procedure ... 30, 39
Set_Inherit_Scheduling
procedure ... 30, 39
Set_Policy
procedure ... 30, 39
Set_Priority
procedure ... 30, 39
Set_Priority_Ceiling
procedure ... 18, 22, 39
Set_Process_Shared
procedure ... 18, 20, 39
Set_Protocol
procedure ... 18, 39
Signal
procedure ... 21, 39
Stacksize_Is_Supported
function ... 12, 39
Stacksize_Support
subtype ... 8, 39
storage_size
attribute ... 8, 39

T

Task_Attributes
type ... 28, 39
Task_ID
type ... 33, 39
Threads_Are_Supported
function ... 12, 39
Threads_Support
subtype ... 8, 39
Timed_Wait
procedure ... 22, 39
Try_Lock
procedure ... 19, 39

U

Unlock
procedure ... 19, 39

V

W

Wait
procedure ... 22, 39

X

Y

Z

Copyright ©1992 E.W. Giering III and T.P. Baker. All rights reserved.
This is an unapproved Draft, subject to change.