Interfacing Ada to Operating Systems

Stephen Michell
Maurya Software Inc
Ottawa, Ontario
stephen.michell@maurya.on.ca

Abstract

This paper examines approaches used by the Ada programming language to interface to explicit operating system services such as events and sockets. We examine the potential for updating a specific interface, the POSIX Ada binding, to a more gereral interface to operating systems services. An approach is proposed to unify classes of services such as synchronous and asynchronous file IO in such an interface.

1.Introduction

The Ada Programming Language has always had a deficiency of interfaces to operating systems. Except for the simple interfaces such as File_IO, most provided interfaces (by the OS) to get services such as Messages, Queues, Events and Asynchronous_IO are not available as Ada packages and subprograms.

A notable exception to this was the POSIX Ada binding[POSIX], which began life as IEEE 1003.5c, and was adapted to Ada 1995 and published as IS14519:2001. Even this interface shows its age as POSIX has moved from a notion of Processes to one that includes Processes and Threads. Ada has added Protected Objects, access to subprograms and protected subprograms, and Interfaces. Collectively, these changes to Ada provide better ways to consider aiding capabilities to an interface such as an interface to an operating system.

We examine the POSIX Ada Binding, and then propose a set of interfaces to generalized operating systems to replace the POSIX Ada Binding, and finally show how new Ada capabilities could be used to integrate diverse concepts such as File_IO and Asynchronous_File_IO.

This paper is organized as follows. Section 2 discusses the layout of the POSIX Ada Interface, and the types of services that it manages. Section 3 proposes a new set of interfaces to operating systems. Section 4 proposes a way to incorporate synchronous and asynchronous behaviours when using operating system calls. Section 5 gives conclusions.

2.The POSIX Ada Binding

The POSIX Ada Binding [APOSIX98] began in the 1980's as an interface to the most popular operating system specification [POSIX]. By its nature, it captured the interface as a set of packages and subprograms to express the functionality. [APOSIX] was updated to incorporate changes to POSIX and Ada and published as IEEE 1003.5c:1998 and IS14519:2001. Since then, the binding has not been updated, even though POSIX has undergone 2 significant revisions , incorporated much of POSIX Real Time, and Ada has incorporated major functionality changes for Ada 2005. Some of the changes undertaken by Ada have a direct impact on the binding by duplicating functionality originally supplied by POSIX with Ada-supplied primitives.

In [WMM2006] we did an analysis on the

existing binding and POSIX and recommended that WG9 begin a revision of the binding.

The POSIXBinding to Ada can be characterized as a set of Ada packages that define subprogram calls to the underlying OS to implement file IO, environment variables, memory management, process scheduling, timers, events, signals, semaphores, and sockets. A summary of the interface packages is in Table.

There has always been a significant tension between POSIX and the Ada runtime system with regards to the management of computational resources such as task scheduling, intertask notifications, and memory. The benefit of using POSIX was that it provided capabilities (different thread scheduling paradigms, timers, and events) that were not available natively in Ada, and that it provided intertask(thread) services to programs that do not have the rich concurrency paradigm of Ada. The challenge has always been that there are significant issues trying to send or receive POSIX signals or events between tasks and POSIX thread, either within a single program or in separate programs. It has almost always been the case that one should not intermix Ada tasks with POSIX threads.

Nevertheless, the POSIX Binding to Ada has been successfully used to interface Ada programs to POSIX. Now that Ada has been updated and has added many of the tasking paradigms that were provided by POSIX, it is reasonable to see if this binding could be replaced by a set of interfaces as part of Ada that are less OS-specific and which better integrate with the Ada runtime.

4. Ada and OS services

We note that [POSIX01] is just one of the operating system interfaces needed in a modern Ada program. There is a set of operating systems called *Windows* [Win95], [Win98], [Win2000,] [Win] that are exceedingly important to Ada programs, as well as embedded systems such as [Wind River]. Argueably, it does not make sense for Ada to attempt to focus its energies on the mapping to a single class of operating systems; rather a strong case can be made to bring the appropriate OS services that have been defined for POSIX, extend them to general purpose OS's and include them in a set of packages in Ada.Interfaces. For example, all functionality related to memory should be in Ada.Interfaces.Memory, such as in Ada.Interfaces.Memory.Locking and in Ada.Interfaces.Memory.Maps.

We therefore propose that instead of updating the POSIX Binding to Ada, that a new set of energies be created to replace them which also handle the general issue of interfacing to the underlying operating system.

Table 2 presents a proposed set of package interfaces to general purpose operating systems. These interfaces roughly correspond to the interfaces from [APOSIX98] but would be recasted to make better use of Ada05 characteristics and lessons learned, such as use of protected operations to specify timing_events as described in [Ada05] section D.15 . An example of one such interface is shown in figure 1.

Capability	**Binding Package**	**Ada Capability**	**Comments**
Page_Alignment	POSIX_Page_Alignment	Nothing	
Environment Varia	POSIX_Process_Environment	Ada.Environment_Variables	
Time Zone	POSIX_Calendar	Ada.Calendar	
Process Primitives	POSIX_Process_Primitives	Nothing, tasks = threads	
Events	POSIX_Signals	Exceptions, entries, prot subprograms and entries	Some thread oriented, others process oriented
Per-thread-timing	POSIX_Process_Times	Ada.Execution_Time	
Process identificati	POSIX_Process_Identification	Ada.Task_ID, Ada.Task_Attributes	
Time	POSIX_Calendar	Ada.Calendar	
File-access	POSIX_permissions		
System Config	POSIX_Configurable_System_Limits	Nothing	
File manipulation	POSIX_Files	Ada.Directories	
	POSIX_File_Status	Ada.Directories	Incomplete
File_IO	POSIX_IO	Ada.Text_IO, Ada.Streams_IO	Ada missing notions of locking and ownership
File Locking	POSIX_File_Locking	Nothing	
Asynchronous_IO	POSIX_Asynchronous_IO	Nothing	
Terminal_IO	POSIX_Terminal_FunctionS	Nothing	
User Information	POSIX_User_Database	Nothing	
Group Information	POSIX_Group_Database	Nothing	
Synchronization	POSIX_Semaphores	Task Entries & POs	
Mutexes	POSIX_Mutexes	Task Entries & POs	
Condition variables	POSIX_Condition_Variables	Task Entries & POs	can be dynamic- POSIX
Memory Locking	POSIX_Memory_Locking	Nothing	
	POSIX_Memory_Range_Locking	Nothing	
Memory_Mapping	POSIX_Memory_Mapping	Nothing	
Shared Memory	POSIX_Shared_Memory_Objects,POSIX_Generic_Shared_Memory	Partitions	
Scheduling	POSIX_Process_Scheduling	Ada Real Time Dispatching	
Timers	POSIX_Timers	Ada.Calendar, Ada.Real_Time.Timing_Events	
Messages	POSIX_Message_Queues	Nothing for process, entries for tasks&POs	
Task Management	Ada's Task_ID		
Interprocess comn	POSIX_XTI	Annex E	
Sockets	POSIX_Sockets	Nothing	
Events	POSIX_Events	Nothing	

Table 1: POSIX Binding capabilities in Ada

Capability	**Ada Capability**	**Ada.Interfaces**
Page_Alignment	Nothing	
Environment Variables	Ada.Environment_Variables	
Time Zone	Ada.Calendar	
Process Primitives	Nothing, tasks = threads	
Events	Exceptions, task entries, protected subprograms and entries	
Per-thread-timing	Ada.Execution_Time	
Process identification	Ada.Task_ID, Ada.Task_Attributes	
Time	Ada.Calendar	
File-access		
System Configuration	Nothing	
File manipulation	Ada.Directories	
	Ada.Directories	
File_Limits		
File_IO	Ada.Sequential_IO,	
Asynchronous_IO	Ada.Direst_IO, Ada.Text_IO, Ada.Streams_IO	File.IO see section ???
File Locking	Nothing	File.Locking
Terminal_IO	Nothing	File.Terminal_IO
User Information	Nothing	Users
Group Information	Nothing	Groups
Data Interchange	Nil	
Synchronization	Task Entries and prot objects	Semaphores
Mutexes	Task Entries and prot objects	Mutexes
Condition variables	Task Entries and prot objects	Condition_Variables
Memory Locking	Nothing	Memory.Locking Memory.Range_Locking
Memory_Mapping	Nothing	Memory.Maps
Shared Memory	Partitions	Memory.Shared_ Memory.Generic_Shared_Memory
Scheduling	Ada Real Time Dispatching	Scheduling.Process_Scheduling Scheduling.Thread_Scheduling
Timers	Ada.Calendar, Ada.Real_Time.Timing_Events	
Messages	Nothing for processes, entries for tasks, protected objects	Message_Queues
Interprocess commun	Annex E	XTI
Sockets	Nothing	Sockets
Events	Nothing	Events

Table 2: Proposed Ada.Interfaces structure for OS Interfacing

```
package Ada.Interfaces.File_IO is

  type File_Type is interface;
  procedure Put(File : File_Type;...);
  ...
  type Asynchronous_Notify is access protected procedure F;
  type Asynchronous_File_Type is interface;
  -- put, get, etc inherited from File_Type
  -- or extended to add capabilities such as
  -- including an access-protected-procedure that
  -- is called when the action completes
  procedure Put(File        : Asynchronous_File_Type;
                IO_Complete : Asynchronous_Notify; ...);
  procedure Wait_For_Completion(File : Asynchronous_File_Type;...);
end Ada.Interfaces.File_IO
```

Figure 1: Example IO interface using Interface types

3. Asynch Task Interfaces

A challenge for applications interfacing Ada programs to operating system services is that these services often provide explicit thread control in the OS while Ada tasking (equivalent to threads) is managed by the language. The challenge for an Ada-OS interface that this view can be quite different since Ada has protected operations, entries, and conditional and timed entries are based on the state of the called task while most OS's use Events, Semaphores, Signals, One-way messages and sockets. Integrating the two views means having to match OS ID's and states and Ada ID's and states explicitly.

For example, OS signals raised by a task or are caught by a task can be synchronous or asynchronous. Ada95 provided mechanisms to notify tasks about events, the protected procedure, and protected entry, but these are asynchronous and are insufficient to map the OS services directly to the language.

Now that Ada 2005 has added access to protected procedures, there are more opportunities to interfaced to OS services than previous versions. Instead of “magic” subprograms that handle an interface, Ada2005 permits us to bridge the concurrency divide between the OS and Ada tasks at protected subprogram boundres. For every signal, event, file_IO, message or socket that requires a concurency-based interface, use protected interfaces and access to protected procedures to capture these events from outside the Ada program and to bring them into the Ada program. Examples of this are provided in the Ada 2005 Rationale [BARNES2006].

Figure 1 shows an outline of a sample package that could integrate synchronous IO and asynchronous IO as part of an interface to operating systems. The obvious parallels can be drawn for Sockets, Events and Message_Queues where a single interface can be provided and both the synchronous interface and a parallel asynchronous interface.

We propose that such interfaces be used in the definition of a collection of services to bind

Ada programs to Operating systems. For example, a task could call Ada.Interfaces.File_IO.Put on File_Type of Asynchronous_IO and return immediately, then call either Wait_For_Completion or call the protected operation that was passed to Put for the parameter IO_Complete. By setting up multiple protected interfaces, a single task can manage multiple asynchronous read or write operations, without the need to explicitly task ID's, into POSIX thread ID's. This approach is usable for many of the POSIX interfaces.

3.Conclusion

This paper has shown an approach to update the POSIX-Ada binding to a general interface to operating systems. It proposes to add these as children of Ada.Interfaces. It also shows an approach that could unify the treatment of synchronous interfaces and asynchronous interfaces using the new Ada *interface* mechanism.

4.Bibliography

[Ada83] ANSI 1815:1983, The Ada Programming Language

[Ada95] ISO/IEC 8652:1995, The Ada Programming Language

[Ada05] ISO/IEC 8652:2007, The Ada Programming Language

[APOSIX] IEEE 103.5c:1998 and IS14519:2001,POSIX Binding to the Ada Programming Langauge

[BARNES2006] Barnes, John, Rationale for Ada 2005, available online from www.adaic.com

[POSIX96] IEEE 1003.1,2,3, and IS9945-1:1996, The Portable Operating System Interface,

[POSIX2003] IEEE 1003.1,2,3, and IS9945-1:2003, The Portable Operating System Interface,

[WMM2007]Wong, Luke, ichell, Stephen, Moore, Brad, Initial Work Scope Summary for updating Ada POSIX Binding IS 143519:2001 to the Ada Programming Language IS8652:2007, available from ISO/IEC/JTC1/SC22/WG9 Ada Working Group as document N477r.