Ada & Embedded/Real Time Linux

Ted Baker
Dept of Computer Science
Florida State University
Tallahassee, FL 32306-4530
USA

baker@cs.fsu.edu
http://www.cs.fsu.edu/~baker

SIGAda

14 November 2000

Acknowledgements

- *Hongfeng Shen – RT Linux V1 GNULLI*
- *Arnaud Charlet – GNAT & GNARL integration, testing, optimizations*
- *Mike Kamrad & Top Layer Networks – GNARL pruning*

Outline

- Motivation: embedded Ada and OS needs
- RT Linux
- A GNAT runtime system for RT Linux
- More Recent Developments
- Future Prospects
- *Maybe some comments on patents**

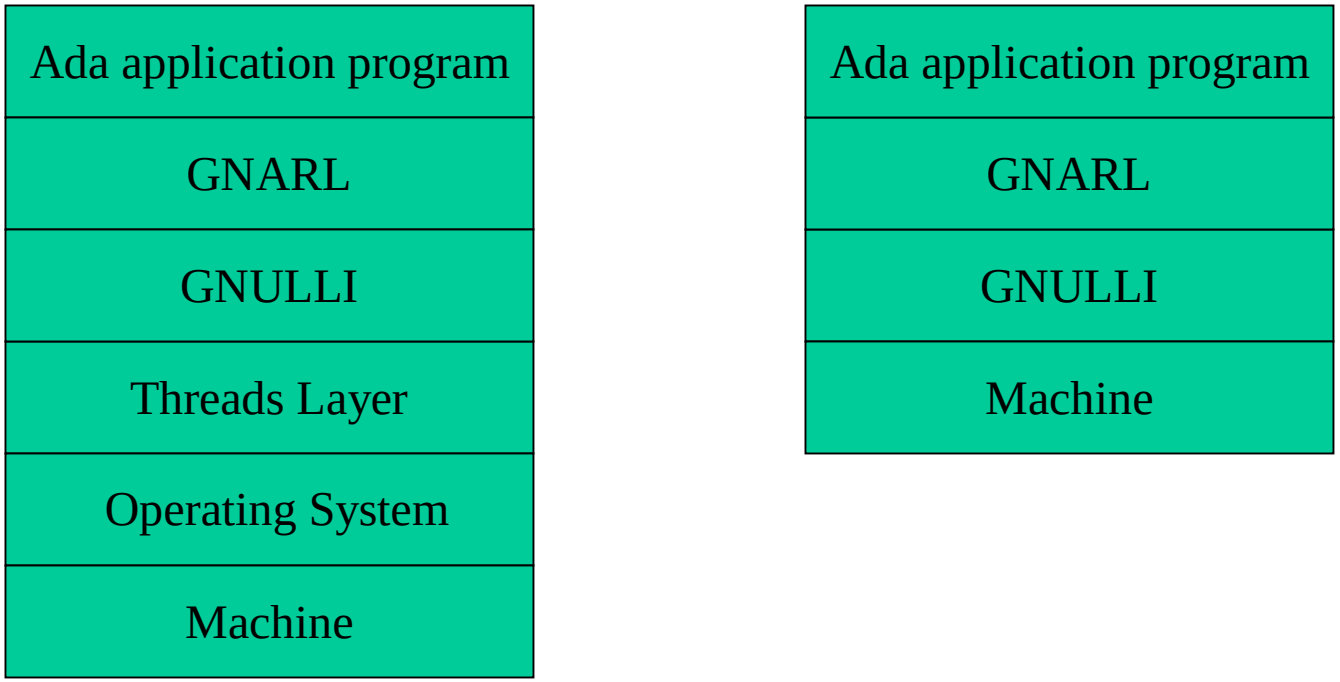
Motivation

- Ada tasking was intended to be implementable on a bare machine
- Ada 95 improved the fit
- Ravenscar profile went further
- Operating systems still limit performance
- Need “free” real-time Ada platform
- Research and instructional tool

Ada 95 Rationale Tasking Model

- Strictly preemptive scheduling
- Queueless locks
- Only ready tasks can hold locks
- Require locker active priority > lock ceiling
- Lock holder inherits ceiling of lock
- Lock operation raises active prio. to ceiling
- Unlock restores previous prio. & schedules

GNARL Architecture



Middle layers can be removed.

The Real-Time OS Problem

- MS Windows & full Unix systems not suitable for hard real time applications
- Cannot be retro-fitted
  - Too large
  - Too complex
  - Too unreliable
- Specialized RT OS's exist…but which one?

Real-Time OS Woes

- Not standardized, despite POSIX efforts
- Proliferation of products, versions, features: Which one(s) should Ada runtime support?
- Conflict of demand for large-OS features *vs.* predictability, reliability, small footprint
- Advanced real-time ideas not being supported quickly enough
- Closed source code

Build a New RT OS?

- OS very costly to develop, maintain
- Device drivers are especially costly
  - Poor documentation of device interfaces
  - Debugging difficulty
  - Plethora of devices need to be covered
- Multiple hardware platforms
- Patents* may restrict new entries
- Who can catch up with big vendors?

A Solution

- Layer real-time OS beneath existing OS
- Provide virtual machine for background OS
- Run real-time tasks in foreground
- Provide data and control connections between foreground and background

Benefits

- Predictable hard-real-time performance
- Complex non-real-time functions can be performed in the background OS
- Background OS is intact
- Background OS device drivers inherited for uses without hard real-time constraints

An early layered kernel: TARTE

- Ada real-time kernel beneath DOS
- DOS runs as background task
- Supports Ada 98 “featherweight tasks”
- Predictable timing, low overhead
- FSU developed for *STARS Foundations*, delivered in 1988

1996 Plan at FSU

- Apply TARTE idea to Ada 95
- Build tasking kernel to run directly on hardware
- Run DOS or Windows 95 in background

Then came RT Linux …

What is RT Linux?

- Small real-time OS that lies under Linux, the popular “free” Unix operating system
- Supports threads inside the Linux kernel

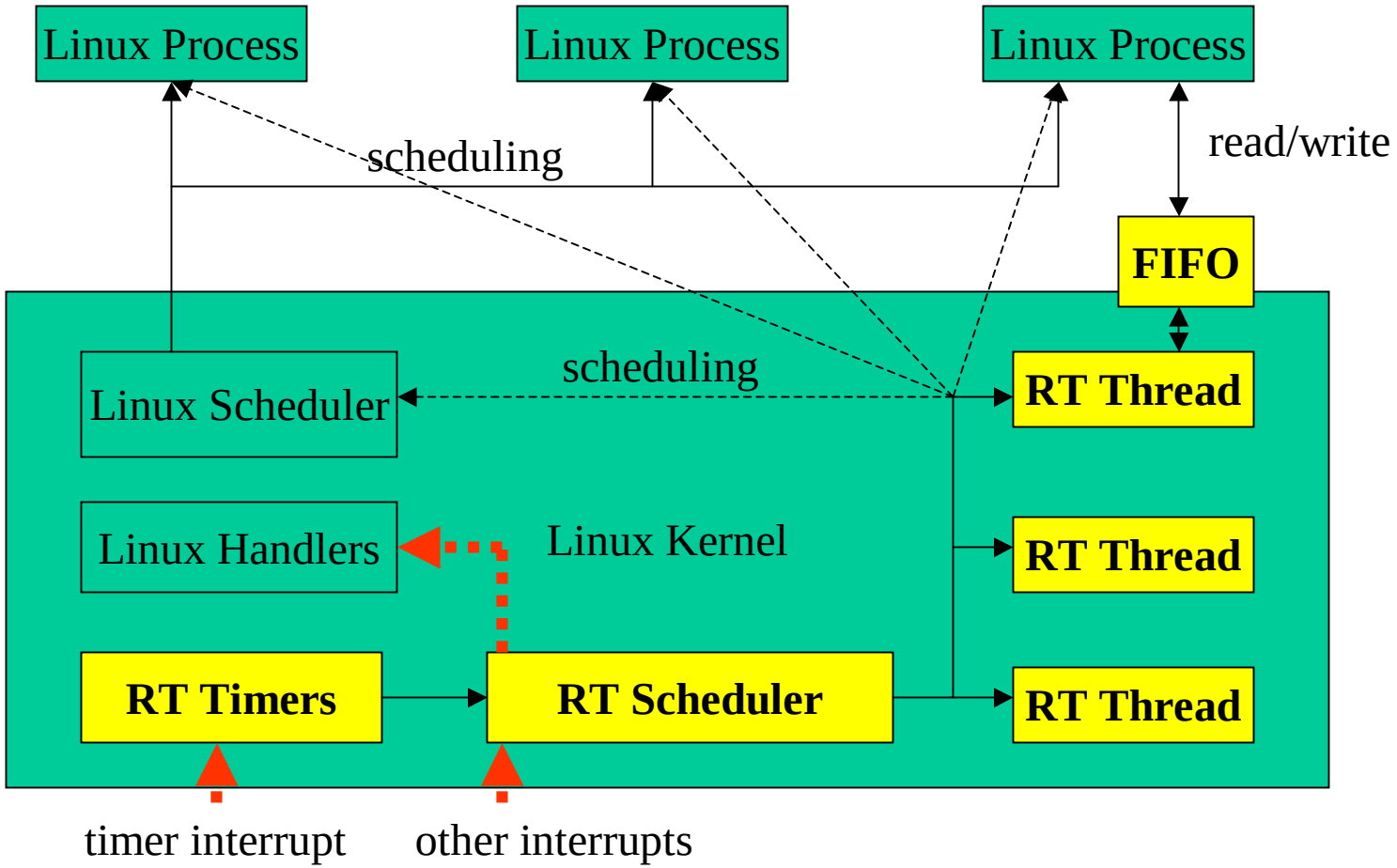
RT Linux™

- Victor Yodaiken, *et al.*
- New Mexico Institute of Technology
- Trademark of FSMLabs
  - www.rtlinux.com
- Open-source software, under GPL
  - www.rtlinux.org
- For Linux kernels 1.0 – 2.4
- For x86, PowerPC, Alpha

RT Linux Features

- Foreground real-time threads
- Shared virtual address space with Linux
- Postponed delivery of hardware interrupts
- FIFO buffers between fore and background
- Easily replaceable scheduler
- Fine-grained clock and interval-timer
- Implemented *via* Linux kernel modules

RT Linux Model



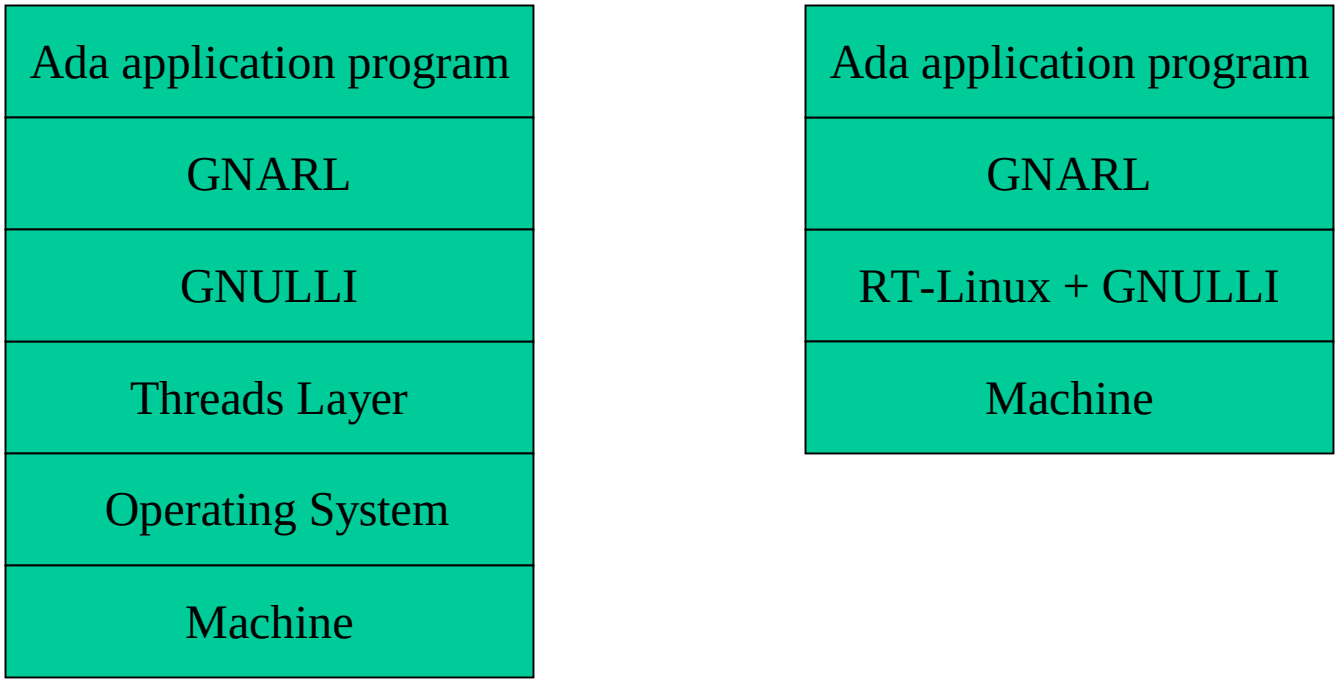
Linux Kernel Modules

- OS extensibility mechanism
- Dynamically loadable and unloadable
- Run in kernel address space
- `init_module` runs when module inserted
- `cleanup_module` when module removed

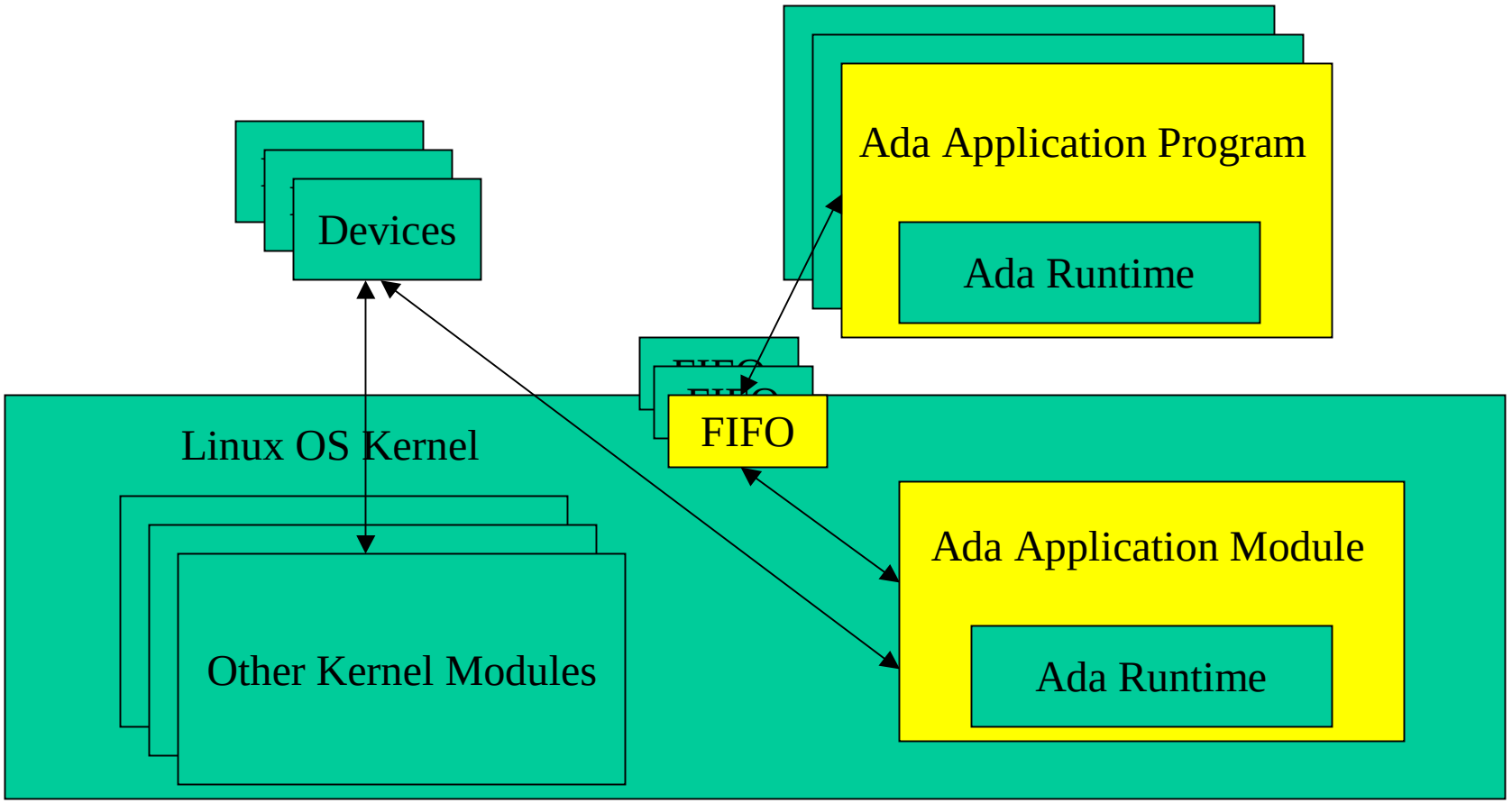
1997 Plan

- Ada application : Linux kernel module
- Port GNULLI to RT Linux
- Test GNULLI directly, then integrate
- Restrict GNARL as necessary
- Modify GNAT to support kernel environ. and optimize for restricted cases

GNARL RT Linux Architecture



Foreground & Background



RT Linux GNULLI

- Implemented in 1997-98
- Based on RTLinux Version 1.0-1.2

  (which supported only single CPU)
- Discarded RTLinux scheduler
- Kept interrupt hijacking, timer, FIFOs

Simplifications

- Single control block per task/thread
- Pre-allocated control blocks
- Pure ceiling locking, no lock queues

Initial Challenges

- No experience with kernel modules in Ada
- Cannot use C header files directly
- GNARL was more than wanted, intertwined
- Need to execute before elaboration
- Running in kernel imposed restrictions

Kernel Restrictions

- Memory is limited
- Modules must be complete
  - May only refer to symbols in kernel & previously loaded modules
- Cannot make OS calls from inside kernel
  - *e.g.*, no *malloc()* or I/O
  - GNAT uses dynamic storage allocation

GNULLI Performance (1999)

- Harmonic task set: 320, 160, 80, 40, 20, 10 Hz
- Pattern:
  - lock; work; unlock; work; lock; work; unlock;
- Work depends on *load level* parameter
- *Load level* is adjusted, by bisection, until maximum schedulable utilization is reached
- 97% CPU utilization

  on 90MHz/33MHz Pentium PC, one CPU

GNULLI Lock-Unlock Performance

1.06 microseconds per cycle

on 90MHz/33MHz Pentium PC, one CPU

Ada Tasking Performance

	FSU threads	**Linux threads**	**RT-Linux threads**
simple protected procedure	2.3 us	3.1 us	1.1 us
protected procedure	3.5 us	4.7 us	1.4 us
po ‘‘rendezvous”	4.6 us	9.7 us	3.1 us

These figures are on 166MHz/66MHz Pentium PC

What we Knew in 1999

- Linux kernel modules can be written in Ada
- Ada tasking supportable in the Linux kernel
- Ravenscar restrictions are helpful, but tighter than necessary
- Timing predictability is as desired
- Overhead is very low

Where we Expected to go

- Get RT Linux port into GNAT distributions
- Improve ease of use
- Further reduce runtime system size
- Add further compiler optimizations
- Add “standard” packages for FIFOs
- Keep up with new versions of RT Linux
- Experiment with sporadic scheduling, dynamic loading, distributed systems annex partitioning

Why we are not there

- No funding for FSU Ada work
- Changes to RT Linux
- New technical issues related to SMP
- *Chairing a growing department*

  www.cs.fsu.edu

Divergent RT Linux Evolution

- Several RTLinux™ versions: 1, 2, 3
- Several more spin-off variants, *e.g.*
  - RTAI
    - www.aero.polimi.it/projects/rtai
  - TimeSys Linux/RT ™
    - www.timesys.com
  - Zentropix Realtime Linux
    - www.zentropix.co.uk

RT Linux Versions

- V1 – the one we targeted
- V2 – the current “stable” version
  - Starts to be like POSIX threads
  - Supports SMP
- V3 – the development version
  - Aims to be closer to POSIX
- Conservative view about adding to API
- Maybe converging

What we are doing at FSU

- Update to RT Linux Version 2
- SMP support
- Single and queue-per CPU models
- Performance comparisons
  - C RTLinux versus Ada
  - Single timer+ready queues versus per-CPU
  - Static versus dynamic queue

New Technical Issues

- New RT Linux API
  - requires GNULLI recoding
- SMP
  - Breaks old GNULLI environment task mapping
  - Opens hard design choices
  - Generally adds complexity

More Issues

- How to recover from failures in kernel
- Continuing order problems
- Getting “self” depends on CPU
  - Per-cpu data is all volatile
  - frequently accessed for spinlocks

Conclusion

- RT Linux V1 GNAT port seems to work, and is in gnat.com baseline, but not in public releases
- RT Linux a valuable platform for Ada
- RT Linux V2 port is not ready yet
- There are some questions left…

THE END

Questions

- Should we revert to Pthreads?
- Is RT Linux the final answer?
- Should we worry about patents?

Should we revert to Pthreads?

- Some more overhead would be paid
- RT Linux is moving that direction
- Portability and upkeep would be reduced
- Usual Gnu leverage
- Yodaiken seems opposed to all priority inversion fixes, including ceiling locking
- Convince him?

Is RT Linux the final answer?

- Safety could be better
- Linux kernel big, complicated, still there
- Protect RT kernel against Linux kernel?
  - Separate virtual address space
- Protect RT kernel against threads?
- Debugging support?
- Patent may get in the way of improvements

Yodaiken Patent

- US Patent 5995745
- Applied for Nov 1997
- Granted Nov 1999
- “Adding real-time support to general purpose operating systems”
- Liberal (free) licensing for Gnu software

Opening the Patent Door Wider

- The Yodaiken RTLinux patent is a peek at a much bigger picture.
- Check out www.uspto.gov/patft.
- Try tracing back the references.
- Many surprising and interesting patents show up.

US Patent 5469571

- Lynx Real-Time Systems, Inc
- Applied July 1991
- Granted Nov 1995
- “Operating system architecture using multiple priority light weight kernel task based interrupt handling”

US Patent 5515538

- Sun Microsystems, Inc.
- Applied March 1994
- Granted May 1996
- “Apparatus and method for interrupt handling in a multi-threaded operating system kernel”

US Patent 5247675

- IBM
- Applied Aug 1991
- Granted Sep 1993
- “Preemptive and non-preemptive scheduling and execution of program threads in a multitasking operating system”

Questions

- Are these patents valid?
- Was there prior art?
- Are the ideas obvious?
- How many more well-known or obvious software ideas are patented or have patents pending?
- Is it safe to write software any more?

APPENDIX

Kernel Module in C

```
#define MODULE
#include <linux/module.h>

int init_module (void)
{ printk (“Hello, World\n”); return 0;}

void cleanup_module (void)
{ printk (“Goodby, World!\n”);}
```

Ada Kernel Module

```
package Hello is
  type Aliased_String is array (Positive range <>)
   of aliased Character;
  pragma Convention (C, Aliased_String);
  Kernel_Version : constant Aliased_String
   := "2.0.33" & Character'Val (0);
  pragma Export (C, Kernel_Version, kernel_version");
  Mod_Use_Count : Integer;
  pragma Export (C, Mod_Use_Count, "mod_use_count_");
  procedure Printk (Message : String);
  pragma Import (C, Printk, "printk");
  function Init_Module return Integer;
  pragma Export (C, Init_Module, "init_module");
  procedure Cleanup_Module;
  pragma Export (C, Cleanup_Module, "cleanup_module");
 end Hello;
```

Ada Kernel Module (body)

```
package body Hello is
  procedure Hello_Elabb;
  pragma Import (Ada, Hello_Elabb, "hello___elabb");
  function Init_Module return Integer is
  begin Hello_Elabb;
    Printk ("Hello, World!" & Character'Val(10));
    return 0;
  end Init_Module;
  procedure Cleanup_Module is
  begin Printk ("Goodbye, World!" & Character'Val(10));
  end Cleanup_Module;
end Hello;
```

GNULLI Operations

- create/destroy task
- lock/unlock
- (timed) sleep/wakeup

Lock Operation on Pthreads

```
procedure Write_Lock
  (L : access Lock; Ceiling_Violation : out Boolean) is
   Result : Interfaces.C.int;
Begin
   Result := pthread_mutex_lock (L.L'Access);
   Ceiling_Violation := Result /= 0;
end Write_Lock;
```

(Assumes mutexes support ceiling locking, which they generally do not.)

Bare Single-CPU Lock Operation

```
procedure Write_Lock
 (L : access Lock; Ceiling_Violation : out Boolean) is
   Prio : constant System.Any_Priority :=
     Current_Task.LL.Active_Prio;
begin … ceiling check omitted …
   L.Pre_Locking_Prio := Prio;
   Current_Task.LL.Active_Prio := L.Ceiling_Prio;
   if Current_Task.LL.Outer_Lock = null then
      Current_Task.LL.Outer_Lock :=
        L.all'Unchecked_Access;
   end if;
end Write_Lock;
```

SMP Lock

- Needs per-CPU data
- Needs Spin_Lock

Unlock Operation on Pthreads

```
procedure Unlock (L : access Lock) is
   Result : Interfaces.C.int;
begin Result := pthread_mutex_unlock (L.L'Access);
end Unlock;
```

Bare Unlock Operation

```
procedure Unlock (L : access Lock) is
   Flags : Integer;
begin
   if Current_Task.LL.Outer_Lock = L.all'Unchecked_Access then
      Current_Task.LL.Active_Priority :=
         Current_Task.LL.Current_Priority;
      Current_Task.LL.Outer_Lock := null;
   else Current_Task.LL.Active_Priority := L.Pre_Locking_Priority;
   end if;
   if Current_Task.LL.Active_Priority <
      Current_Task.LL.Succ.LL.Active_Priority then
      Save_Flags (Flags); -- Saves interrupt mask
      Cli; -- Masks interrupts
      Delete_From_Ready_Queue(Current_Task);
      Insert_In_Ready_Queue (Current_Task);
      Restore_Flags (Flags);
      Call_Scheduler;
    end if;
end Unlock;
```

SMP Unlock

- Needs CPU-specific data
- Needs Spin_Unlock
- Also needs spinlock to protect ready queue, for priority-lowering
- Needs IPI, with single-queue model

Sleep on Pthreads

```
procedure Sleep (Self_ID : Task_ID; Reason : Task_States) is
   Result : Interfaces.C.int;
Begin
   if Self_ID.Pending_Priority_Change then
      Self_ID.Pending_Priority_Change := False;
      Self_ID.Base_Priority := Self_ID.New_Base_Priority;
      Set_Priority (Self_ID, Self_ID.Base_Priority);
   end if;
   Result := pthread_cond_wait
     (Self_ID.LL.CV'Access, Self_ID.LL.L.L'Access);
   pragma Assert (Result = 0 or else Result = EINTR);
end Sleep;
```

Bare Sleep

```
procedure Sleep
  (Self_ID : Task_ID; Reason : System.Tasking.Task_States) is
   Flags : Integer;
begin Self_ID.State := Reason
   Save_Flags (Flags); Cli;
   Delete_From_Ready_Queue (Self_ID);
   if Self_ID.LL.Outer_Lock = Self_ID.LL.L'Access then
      Self_ID.LL.Active_Prio := Self_ID.LL.Current_Prio;
      Self_ID.LL.Outer_Lock := null;
   else
      Self_ID.LL.Active_Prio := Self_ID.LL.L.Pre_Locking_Prio;
   end if;
   Restore_Flags (Flags); Call_Scheduler;
   Write_Lock (Self_ID);
end Sleep;
```

SMP Sleep

- Needs Spin_Lock on ready queue
- Needs Spin_Unlock on own lock

Wakeup on Pthreads

```
procedure Wakeup (T : Task_ID; Reason : Task_States) is
   Result : Interfaces.C.int;
Begin
   Result := pthread_cond_signal (T.LL.CV'Access);
   pragma Assert (Result = 0);
end Wakeup;
```

Bare Wakeup

```
procedure Wakeup
  (T : Task_ID; Reason : System.Tasking.Task_States) is
   Flags : Integer;
begin T.State := Reason; Save_Flags (Flags);
   Cli; -- Disable interrupts.
   if Timer_Queue.LL.Succ = T then
      if T.LL.Succ = Timer_Queue then No_Timer;
      else Set_Timer (T.LL.Succ.LL.Resume_Time);
      end if;
   end if;
   Delete_From_Timer_Queue (T); Insert_In_Ready_Queue (T);
   Restore_Flags (Flags); Call_Scheduler;
end Wakeup;
```

SMP Wakeup

- Needs Spin_Lock on ready queue
- Needs IPI, with single-queue model

Hidden Init/Cleanup_Module

```
pragma Suppress (All_Checks);
package Demo is
   pragma Elaborate_Body;
end Demo;
```

Hidden Init/Cleanup Module

```
pragma Suppress (All_Checks);
with System.OS_Interface;
package body Demo is
   use System.OS_Interface;
   task T;
   task body T is
   begin
      Printk("Hello, World!” & ASCII.LF);
   end T;
end Demo;
```

Interval vs. Periodic Timer