A POSIX-Ada Interface for Application-Defined Scheduling

By: Mario Aldea Rivas  
Michael González Harbour

(aldeam@unican.es)  
(mgh@unican.es)

Ada-Europe’2002
Vienna, Austria, June 17-21, 2002
Table of Contents

1. Overview
2. Motivation
3. Some Requirements
4. Model Description
5. Scheduling Events
6. Scheduling Actions
7. Example of an Application-Defined Policy: EDF
8. MaRTE OS
9. Performance
10. Conclusions and Further Work
1. Overview

Interface for Application-Defined Scheduling

Integrated into the POSIX-Ada bindings
- POSIX: Portable Operating System Interface standard

Not only an interface
- Requires support in the operating system kernel
- Implemented in our operating system MaRTE OS

Main points
- Application-defined scheduling policies implemented by a special kind of tasks (*scheduler tasks*)
- Scheduler tasks can activate or suspend other tasks
2. Motivation

The scheduling policies defined in Ada and POSIX are not sufficient for all application environments

• Dynamic priorities allow better use of resources
• Dynamic systems require flexible scheduling schemes (multimedia)

There are many policies based on dynamic priorities

• It is not possible to standardize them all

Our proposal:

• Allow applications to define their own scheduling policies and synchronization protocols
3. Some Requirements

- Allow a large variety of scheduling policies
- Compatibility with current scheduling policies in POSIX and Ada
- Coexistence of several scheduling algorithms
- Isolation of critical tasks from behavior of application schedulers
- Integration of synchronization protocols to avoid priority inversion or similar effects
4. Model Description

- **Application-Scheduled Task**
- **System Scheduler**
- **User Address Space**
- **Regular Task**
- **Application Scheduler Task**
- **Scheduler Address Space**
Model Description (Cont’d)

• Scheduler tasks are always invoked before their scheduled tasks
  - They inherit all the priorities inherited by them
  - Schedulers have precedence if same priority
• Schedulers can activate or suspend any of their tasks, and must accept (or reject) them at creation
• They are notified about events related to their scheduled tasks
Model Description (Cont’d)
5. Scheduling Events

They represent situations relevant to the application scheduler

They contain the following information

- Type of event
- Task that caused the event
- Event-dependent information:
  - Inherited priority
  - Involved application mutex
  - Signal received
  - Application-specific information

Events may be filtered by type
## Scheduling Events (Cont’d)

<table>
<thead>
<tr>
<th>Scheduling event type</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW_TASK</td>
<td>none</td>
</tr>
<tr>
<td>TERMINATE_TASK</td>
<td>none</td>
</tr>
<tr>
<td>READY</td>
<td>none</td>
</tr>
<tr>
<td>BLOCK</td>
<td>none</td>
</tr>
<tr>
<td>YIELD</td>
<td>none</td>
</tr>
<tr>
<td>SIGNAL</td>
<td>POSIX signal information</td>
</tr>
<tr>
<td>CHANGE_SCHED_PARAM</td>
<td>none</td>
</tr>
<tr>
<td>EXPLICIT_CALL</td>
<td>Application message</td>
</tr>
<tr>
<td>TIMEOUT</td>
<td>none</td>
</tr>
</tbody>
</table>
## Scheduling Events (Cont’d)

<table>
<thead>
<tr>
<th>Scheduling event type</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIORITY_INHERIT</td>
<td>Inherited system priority</td>
</tr>
<tr>
<td>PRIORITY_UNINHERIT</td>
<td>Uninherited system priority</td>
</tr>
<tr>
<td>INIT_MUTEX</td>
<td>Pointer to the app. mutex</td>
</tr>
<tr>
<td>DESTROY_MUTEX</td>
<td>Pointer to the app. mutex</td>
</tr>
<tr>
<td>LOCK_MUTEX</td>
<td>Pointer to the app. mutex</td>
</tr>
<tr>
<td>TRYLOCK_MUTEX</td>
<td>Pointer to the app. mutex</td>
</tr>
<tr>
<td>UNLOCK_MUTEX</td>
<td>Pointer to the app. mutex</td>
</tr>
<tr>
<td>BLOCK_AT_MUTEX</td>
<td>Pointer to the app. mutex</td>
</tr>
<tr>
<td>CHANGE_MUTEX_SCHED_PARAM</td>
<td>Pointer to the app. mutex</td>
</tr>
</tbody>
</table>
6. Scheduling Actions

Schedulers have an operation (\texttt{Execute\_Actions}) to request execution of multiple actions, including:

- accept or reject a task that has requested attachment
- activate or suspend a task
- accept or reject initialization of an application mutex
- grant the lock on an application mutex

As part of \texttt{Execute\_Actions}, the scheduler suspends until:

- the next scheduling event,
- arrival of a POSIX signal (usually indicating expiration of regular or CPU-time timers)
- or the expiration of a timeout
Execute Actions

```plaintext
procedure Execute_Actions
(Sched_Actions : in Scheduling_Actions;
 Set           : in POSIX_Signals.Signal_Set;
 Event         : out Scheduling_Event;
 Timeout       : in POSIX.Timespec;
 Current_Time  : out POSIX.Timespec)
-- other overloaded versions exist
```

Structure of a scheduler:

1. Execute actions and wait for event
2. Determine actions to execute according to received event
7. Example of an Application-Defined Policy: EDF

Application scheduled task:

```pascal
with POSIX_Application_Scheduling;
package AppSched renames POSIX_Application_Scheduling;
...
package EDF_Policy is new
  AppSched.Application_Defined_Policy (EDF_Parameters);
...

task body Periodic_EDF_Task is
  Param : EDF_Parameters := (Deadline, Period);
begins
  EDF_Policy.Change_Task_Policy (EDF_Scheduler_Task_Id, Param);
  loop
    -- do useful work
    ...;
    -- task will wait until the next period
    AppSched.Invoke_Scheduler;
  end loop;
end Periodic_EDF_Task;
```
Example of an Application-Defined Policy: EDF (Cont’d)

Application Scheduler task:

```vhdl
task body EDF_Scheduler is
    ...;
    Most_Urgent_Task : Task_Id;
    Now, Earliest_Start : POSIX.Timespec;
    Event : AppSched.Scheduling_Event;
    Sched_Actions : AppSched.Scheduling_Actions;
begin
    AppSched.Become_An_Application_Scheduler;
    AppSched.Set_Clock(Clock_Realtime);
    AppSched.Set_Flags(Absolute_Timeout);
    Now := POSIX_Timers.Get_Time(Clock_Realtime);
    -- Scheduling events processing loop
    loop
        Schedule_Next(Most_Urgent_Task, Earliest_Start);
    end loop;
end EDF_Scheduler;
```
Example of an Application-Defined policy: EDF (Cont’d)

```plaintext
if Most_Urgent_Task /= Null_Task_Id then
    AppSched.Add_Activate (Sched_Actions, Most_Urgent_Task);
end case;
AppSched.Execute_Actions_with_Timeout
    (Sched_Actions, Event, Earliest_Start);

-- process scheduling events
case AppSched.Get_Event_Code(Event) is

    when New_Task =>
        if Schedulability_Test (AppSched.Get_Task(Event)) then
            AppSched.Add_Accept (Sched_Actions,
                                AppSched.Get_Task(Event));
            Add_To_List_of_Tasks (AppSched.Get_Task(Event), Now);
        else
            AppSched.Add_Reject (Sched_Actions,
                                AppSched.Get_Task(Event));
        end if;

    when Terminate_Task =>
        Eliminate_From_List_of_Tasks (AppSched.Get_Task(Event));
```

Example of an Application-Defined policy: EDF (Cont’d)

```plaintext
when Ready =>
    Make_Active (AppSched.Get_Task (Event));

when Block =>
    Make_Blocked (AppSched.Get_Task (Event));

when Explicit_Call =>
    AppSched.Add_Suspend (Sched_Actions,
        AppSched.Get_Task (Event));
    Make_Timed (AppSched.Get_Task (Event));

when Timeout =>
    null;

when Others =>  null;
end case;

end loop; -- Scheduling events processing loop

end EDF_Scheduler;
```
MaRTE OS

Minimal Real-Time OS for Embedded Applications

- Follows the POSIX.13 Minimum Realtime System Profile
  - Single process
  - No file system
- Adds new POSIX interfaces: CPU-time Timers
- Written in Ada, with few parts in C or assembly language
- Gnat run-time system runs on top of it
- POSIX threads interface for C threads
- Usable both for C and Ada (and mixed) applications
- Free software

http://marte.unican.es
## Context switches in Ada

<table>
<thead>
<tr>
<th>Scheduling Algorithm</th>
<th>Context switch (μs)</th>
<th>Overhead for a 1KHz task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular Ada delay until statement</td>
<td>5</td>
<td>1%</td>
</tr>
<tr>
<td>Round robin within priorities</td>
<td>4.5</td>
<td>0.9%</td>
</tr>
<tr>
<td>Constant Bandwidth Server</td>
<td>6</td>
<td>1.2%</td>
</tr>
</tbody>
</table>
Conclusions and Further Work

APIs for application scheduling in Ada and C

- Performance is acceptable
- Great flexibility
- Compatible with previous fixed priority scheduling
- Implemented in MaRTE OS

Future work

- POSIX standardization
- Integration into the Ada language
  - will require some language extensions (pragmas, ...), specially for protected objects