

POSIX-Compatible Application-Defined Scheduling

By: **Mario Aldea Rivas** (aldeam@unican.es)
Michael González Harbour (mgh@unican.es)

The Open Group Conference
Boston, July, 2002

Funded in part by FIRST project (*EU project IST2001-34140*)

Table of Contents

- 1. Overview**
- 2. Motivation and Background**
- 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 threads model

- **POSIX: Portable Operating System Interface standard**

Not just a library

- **Requires support by the operating system kernel**

Main points

- **Application-defined scheduling policies implemented by a special kind of threads (*scheduler threads*)**
- **Scheduler threads can activate or suspend other threads**
- **Reference implementation in MaRTE OS**

2. Motivation

The scheduling policies defined in 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**

Background

Many systems provide similar functionality

- **Kernel-modules**
 - **Shark, RT-Linux, Vassal**
- **Non-kernel schedulers**
 - **RED-Linux, CPU-Inheritance**

Kernel modules are difficult to standardize

- **Kernel programming models differ, and reliability is an issue**

We choose the hierarchical scheduler approach

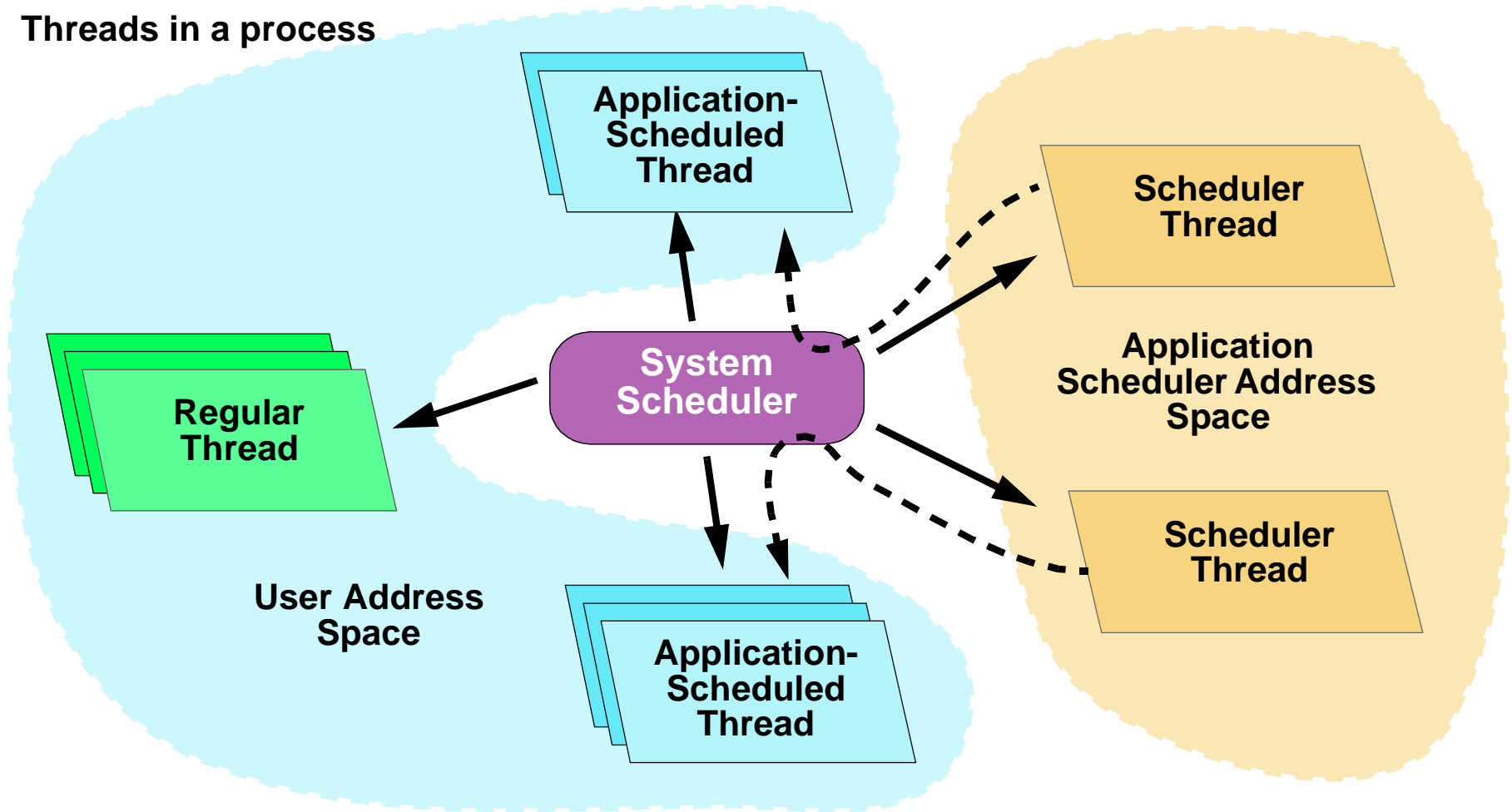
- **Fixed priorities at the root of the hierarchy**
- **Application-defined schedulers on top**

3. Some Requirements

- **Allow a large variety of scheduling policies**
- **Compatibility with current scheduling policies in POSIX**
- **Coexistence of several scheduling algorithms**
- **Isolation of critical threads from behavior of application schedulers**
- **Integration of synchronization protocols to avoid priority inversion or similar effects**

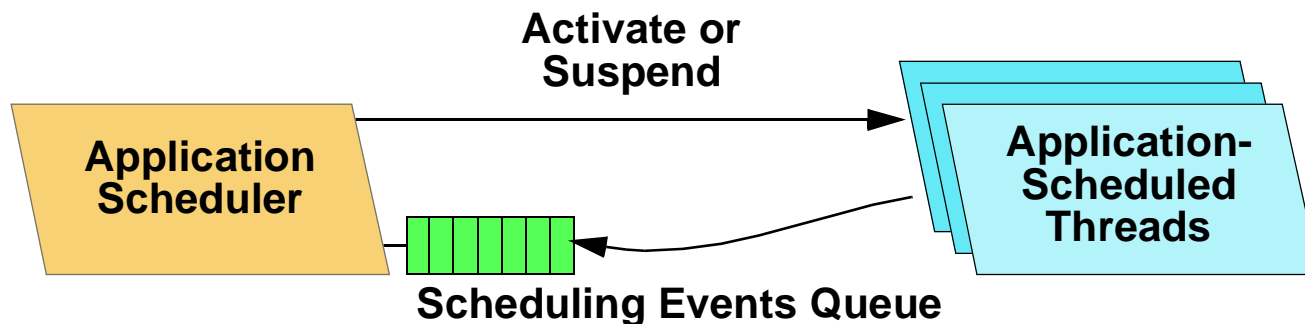
4. Model Description

Threads in a process

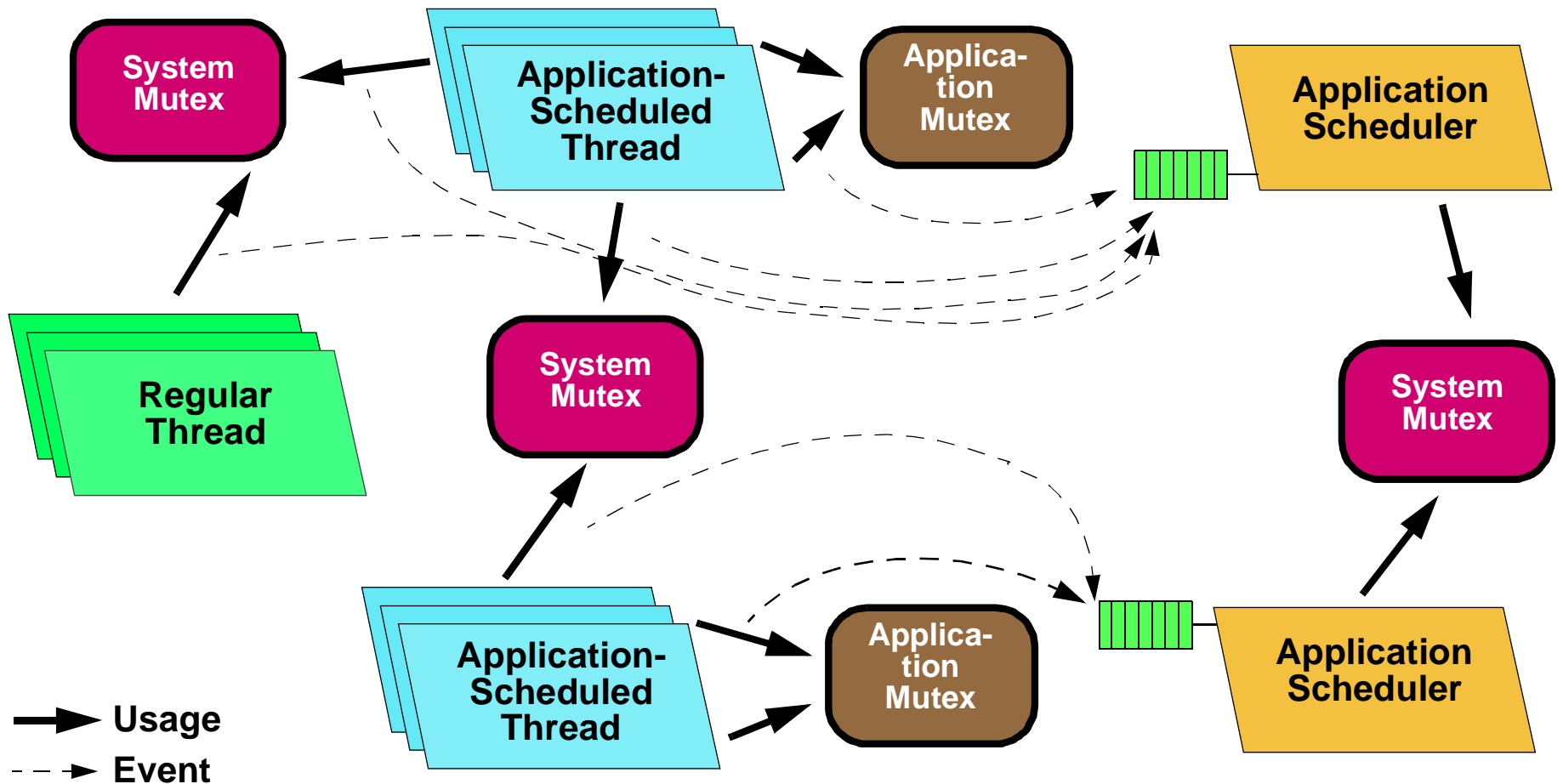


Model Description (Cont'd)

- **Scheduler threads:**
 - are always invoked before their scheduled threads
 - inherit all the priorities inherited by their threads
 - have precedence if same priority
- **Schedulers can activate or suspend any of their threads, and must accept (or reject) them at creation**
- **They are notified about events related to their scheduled threads**



Model Description (Cont'd)



5. Scheduling Events

They represent situations relevant to the application scheduler

They contain the following information:

- **type of event**
- **thread 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)

Scheduling event type	Additional information
NEW_THREAD	none
TERMINATE_THREAD	none
READY	none
BLOCK	none
YIELD	none
SIGNAL	POSIX signal information
CHANGE_SCHED_PARAM	none
EXPLICIT_CALL	User event code
EXPLICIT_CALL_WITH_DATA	Application message
TIMEOUT	none

Scheduling Events (Cont'd)

Scheduling event type	Additional information
PRIORITY_INHERIT	Inherited system priority
PRIORITY_UNINHERIT	Uninherited system priority
INIT_MUTEX	Pointer to the app. mutex
DESTROY_MUTEX	Pointer to the app. mutex
LOCK_MUTEX_REQ	Pointer to the app. mutex
TRYLOCK_MUTEX_REQ	Pointer to the app. mutex
UNLOCK_MUTEX	Pointer to the app. mutex
BLOCK_AT_MUTEX	Pointer to the app. mutex
CHANGE_MUTEX_SCHED_PARAM	Pointer to the app. mutex

6. Scheduling Actions

Schedulers have an operation (`posix_appsched_execute_actions`) to request execution of multiple actions, including:

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

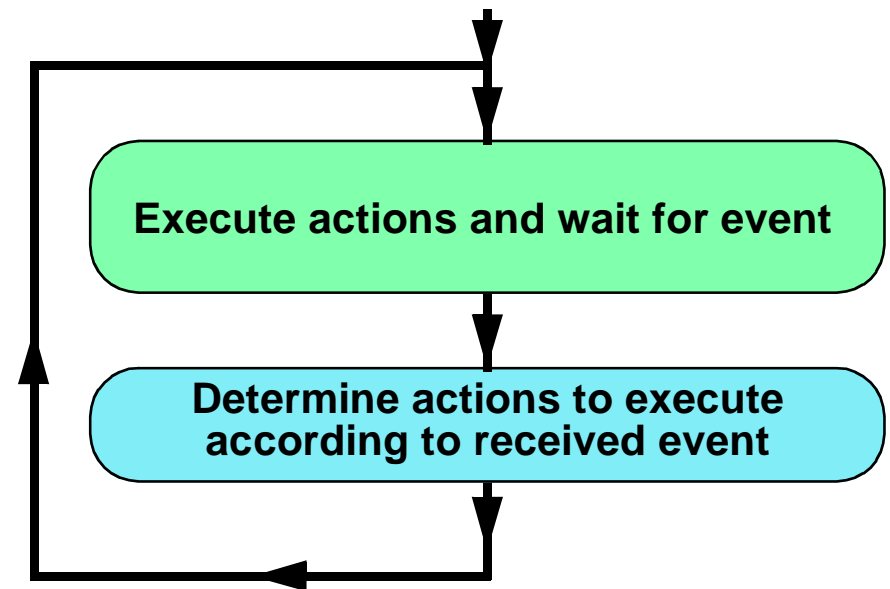
As part of this operation, the scheduler suspends itself until:

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

Interfaces for the scheduler: Execute Actions

```
int posix_appsched_execute_actions
(const posix_appsched_actions_t *sched_actions,
 const sigset_t *set,
 const struct timespec *timeout,
 struct timespec *current_time,
 struct posix_appsched_event *event);
```

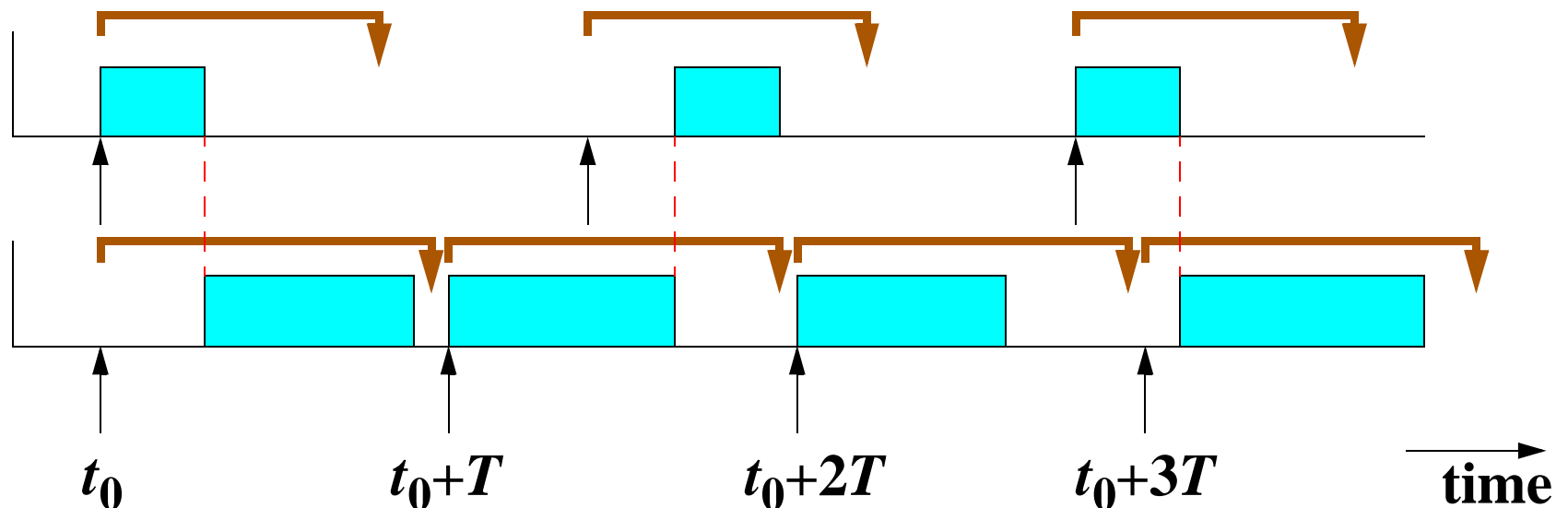
Structure of a scheduler:



7. Example of an Application-Defined Policy: EDF

Earliest Deadline First Scheduling Policy (EDF)

- Schedules periodic threads (period= T)
- Each thread has a relative deadline= d
- Each thread instance has an absolute deadline= $t_0 + nT + d$



Example (Cont'd)

Scheduling parameters type:

```
typedef struct {  
    int edf_thread;  
    struct timespec deadline, period;  
} edf_parameters_t;
```

Application-scheduled thread:

```
void * edf_thread (void * arg) {  
    while (1) {  
        // do useful work  
        ...  
        // tell the scheduler that the current job has finished  
        pthread_appsched_invoke_scheduler (0);  
    }  
}
```

EDF Scheduler Implementation

States of an EDF periodic thread:

- active or waiting

Two priority queues:

- waiting threads, ordered according to activation time
- running threads, ordered according to deadline

The `schedule_next` operation

- switches to the active state the threads that need it
- calculates next thread to be run
- calculates earliest start of waiting threads

Example (Cont'd)

Application Scheduler thread:

```
void *edf_scheduler (void *arg)
{
    initialization;
    while (1) {
        schedule_next (&next_thread, &earliest_start, // out params
                      &now);                          // in params
        /* Thread activation and suspension actions*/
        posix_appsched_actions_addactivate (&actions, next_thread);
        posix_appsched_actions_addsuspend (&actions, current_thread);
        current_thread = next_thread;
        /* Execute scheduling actions */
        posix_appsched_execute_actions
            (&actions, &awaited_signal_set, &earliest_start, // in params
             &now, &sched_event);                          // out params
    }
}
```

Example (Cont'd)

```
/* Process scheduling events */
switch (sched_event.event_code) {
case POSIX_APPSCHED_NEW :
    Get thread EDF specific sched param;
    posix_appsched_actions_addaccept
        (&actions, sched_event.thread);
    Add thread to list of scheduled threads;
    break;
case POSIX_APPSCHED_TERMINATE :
    Remove thread from scheduled threads list; break;
case POSIX_APPSCHED_EXPLICIT_CALL :
    thread.state=WAITING;
    Obtain next deadline & activation time; break;
case POSIX_APPSCHED_TIMEOUT :
    break; // threads will be rescheduled
} // switch
} // while (1)
}
```

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
- Usable both for C and Ada (and mixed) applications
 - implements POSIX threads interface for C threads
- Free software (GPL)

<http://marte.unican.es>

Performance on a 1.1GHz Pentium III running MaRTE OS

Policy: EDF + Constant Bandwidth Server (CBS)

Context switch from a CBS thread that consumed its CPU-time budget, to an EDF thread

Description	Time (μ s)
Scheduler activation after CPU-timer expiration	1.8
Scheduling algorithm (CBS)	1.3
EDF thread activation	1.0
Total context switch time	4.1
Context switch after timer expiration (FIFO scheduling)	1.1

Conclusions and Further Work

APIs developed for application scheduling in POSIX

- Performance is acceptable
- Very flexible
 - every RT policy that we know about can be implemented
- Compatible with previous fixed-priority scheduling
- Implemented in MaRTE OS
- C and Ada interfaces developed

Future work

- POSIX standardization
- Integration into the Ada language