

# Table of Contents

<b>1. Execution Time Timers .....</b>	<b>1</b>
1.1. Description.....	1
1.2. Author .....	1
1.3. Layer .....	1
1.4. API / Compatibility .....	1
1.5. Dependencies .....	2
1.6. Status.....	2
1.7. Implementation issues.....	2
1.8. Validation criteria.....	3
1.9. Tests .....	4
1.10. Installation.....	5

# Chapter 1. Execution Time Timers

## 1.1. Description

When performing a schedulability test, scheduling analysis relies on a known worst-case execution time (WCET). Nevertheless, estimating WCET is a difficult task due to the different execution paths within a program and today's computer architectures. Specially, in the context of concurrent programs in which cache misses are frequent after interrupt service routines or context switches.

Unfortunately the tasking model of most concurrent hard real-time systems, not enforces the bound on the execution time of tasks. Without bound on execution time, a task could execute more than estimated, causing other tasks to loose its deadlines. This, on most hard real time application, may result in catastrophic consequences.

This component tries to give a solution to this problem, implementing execution time timers within the task scheduler. This kind of timers may be used to detect execution time overruns in the application, and to limit their effects.

If a timer is created using a CPU-Time clock of a particular thread, and a relative expiration time is given, it can be used to notify that a certain budget of execution time has elapsed, for that thread. If the timer is armed each time a thread is activated, and the relative expiration time is set to the thread's estimated worst-case execution time (plus some small amount to take into account the limited resolution and precision of the CPU-Time clock), then the timer will only expire if the thread suffers an execution time overrun.

## 1.2. Author

Josep Vidal Canet [jvidal@disca.upv.es](mailto:jvidal@disca.upv.es)

## 1.3. Layer

Low level.

## 1.4. API / Compatibility

The execution time clocks interface defined in the proposed standard POSIX.1d is based on the POSIX.1b clocks and timers interface used for normal real time clocks. The new interface creates a new function to access the execution time clock identifier of the desired thread: `pthread_getcpuclockid()`. In addition, it defines a new thread-creation attribute, called `cpu_clock_requirement`, which allows the application to enable or disable the use of the execution time clock of a thread, at the time of its creation. Once the thread is created, this attribute cannot be modified. To use CPU-time clocks for threads, we must set the `cpu_clock_requirement` attribute to the value `CLOCK_REQUIRED_FOR_THREAD`.

```
#include <time.h>
```

```
int pthread_getcpuclockid(pthread_t *thread, clockid_t *clock_
```

## 1.5. Dependencies

Depends on `psignals` and `ptimers` components.

## 1.6. Status

Testing.

## 1.7. Implementation issues

POSIX defines the execution time as the time spent executing a process or thread, including the time spent executing system services on behalf of that thread. Due to the fact that in RTLinux all threads run in kernel space, system calls are implemented as simple function calls fully executed in the context of the calling thread. This OS characteristic allows to implement CPU-Time execution timers in a very simple and efficient way.

The implementation of CPU-Time clocks and timers in RTLinux requires modification of the data structure that defines each thread, the thread control block,

modifications the scheduler code to include the necessary steps to update each thread's CPU-Time clock and modifications to the timers management code to operate with the CPU-Time clocks associated timers.

The information that has been added to the thread control block consists of:

- A structure with the information needed for the CPU-Time clock, including the clock identifier and the total CPU-Time consumed by that thread.
- A high resolution time type (long long) storing the time of the last activation

The main modifications required to support CPU-Time clocks and timers are focused on the scheduler API and POSIX timers API.

RTLinux CPU clocks implementation follows RTLinux standar way of adding system clocks. Modifications required to the scheduler in order to support CPU clocks consists of adding code at the point where a new thread becomes the running thread and to the function that finds next preemptor thread.

The point where a new thread becomes the running thread in RTLinux is located at the scheduler function (`int rtl_schedule()`) just before the context switch is performed. At this point, what is done is to store the execution time of the current thread and anotate the activation time of the thread that is going to take the CPU.

Finally, the find preemptor function, has modified to handle the situation where the thread that is going to be executed has an execution time timer armed. This modification consists in programing the scheduler timer to shot at the time in which the execution time timer expires (of course, if there isn't any higher priority thread that will preempt it earlier).

## **1.8. Validation criteria**

This OS facility allows to implement today's scheduling algorithms such as CBS and sporadic server schedulers in a more efficent and reliable way , specially at application-level with the use of POSIX-Compatible Application-defined Scheduling (available in RTLinux as an OCERA component).

As said before, execution time timers may be used to increase the fault-tolerance of critical applications due to a system timing malfunction not detected by the off-line analysis techniques.

Finally, the use execution time timers could help to estimate WCET (Worst Case Execution Time) based on statistical analysis.

As in most of the low level components, the implementation of execution time timers tries to fit two objectives: minimize the overhead introduced in RTLinux runtime and try to achieve the best efficiency. In addition to this, an acceptable timer resolution respect the available hardware should be reached.

The overhead introduced to RTLinux runtime due to the use of CPU clocks is negligible (less than 0.01 %) as we will see in test section. Execution time timers precision is near to a few microseconds, as in normal POSIX timers. This is due to the fact that the low level system timer is the same for both.

Finally, execution time timers implementation guarantees the immediacy of the timer expiration notification (similar to a context switch).

## 1.9. Tests

Planned tests have been designed to check CPU clocks and execution time timers RTLinux implementation correctness, accuracy and overhead. From the Open Posix Test Suite project, external tests have been adapted to test CPU clocks implementation. Unfortunately no external tests suites have been found to test execution time timers.

- 

The POSIX Test Suite is an open source test suite with the goal of performing conformance, functional, and stress testing of the IEEE 1003.1-2001 System Interfaces specification in a manner that is agnostic to any given implementation.

Among other POSIX functionalities, these suite support CPU clocks testing. We have used some of this tests slightly modified to run on RTLinux.

- 

To measure the overhead introduced by CPU clocks, the Baker utilization has been passed.

This test allows to measure runtime overhead scheduling six harmonic tasks. Harmonic tasks have the following periods: 1/320HZ, 2/320HZ, 4/320HZ, 8/320HZ, 16/320HZ and 32/320HZ= 100 milliseconds respectively. In each test iteration (every second) tasks load (CPU consumption) is increased by an amount. Test finishes when a task loses its deadline. At this point, the Utilization is calculated taking tasks load from previous iteration.

From this test results, it could be stated that the overhead introduced when using CPU clocks is negligible (less than 0.01 %).

- 

Timers accuracy tests developed while implementing POSIX timers have been adapted to measure execution time timers precision. This tests shows that execution time timers precision is the same as normal POSIX timers. This is due to the fact that the low level system timer is the same for all clocks. This precision is near to the microsecond and depends on the available hardware.

- 

Basic CPU timers functionality has been tested with self built tests.

Among other tests, there is a simple one that shows how the use of execution time timers helps to increase the fault-tolerance of applications due to timing malfunctions. In this tests a execution time timer is armed to expire after the highest priority task execution time reaches one second. At that momenent the execution time timer handler suspends the task in order to allow other tasks to take the CPU.

On the other hand, the highest priority task is doing and endless loop wasting time. If the execution time timer handler doesn't suspends wasting time task, Linux will hang and we will loose control over the computer.

## 1.10. Installation

The code is distributed as a patch file against rtlinux-3.1 source code plus some new files added (do, examples, execution time timers implementation, ..)

In order to install, please follow next steps:

- 

- 1.- Install POSIX signals component.

- 

- 2.- Install POSIX timers component:

- - 3.- Install execution time timers component:
    - - 3.1- Configure Makefile variables: edit Makefile and set RTLINUX variable to your RTLinux copy path.
    - - 3.2.- Type make install
- - 4.- Change to the RTLinux directory.
    - - type make xconfig
    - - enable both posix signals, timers and execution time tmers.
    - - type make clean ; make

New features are by default disabled, you have to enable them before building RTLinux. For more info see also posix signals and timers documentation.