WIKIPEDIA

Rodos (operating system)

Rodos (**R**ealtime **O**nboard **D**ependable **O**perating **S**ystem) is a real-time operating system for embedded systems and was designed for application domains demanding high dependability.

Rodos

Rodos logo

Developer	German Aerospace Center
Written in	C, C++ and Assembly language
Source model	Open source
Platforms	See #Supported Architectures
License	BSD license
Official website	Information & Download (http://www8.informatik.uni-wuerzburg.de/wissenschaftforschung/rodos/)

Contents

History

Rodos was developed at the German Aerospace Center and has its roots in the operating system BOSS. It is used for the current micro satellite program of the German Aerospace Center. The system runs on the operational satellite TET-1 and will be used for the currently developed satellite BiROS.

Rodos is further enhanced and extended at the German Aerospace Center as well as the department for aerospace information technology at the University of Würzburg.

Features

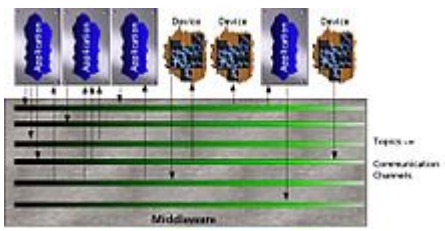
An important aspect of Rodos is its integrated real time middleware. Developing the control and payload software on the top of a middleware provides the maximum of modularity today. Applications/modules can be developed independently and it is very simple to interchange modules later without worrying about side effects, because all modules are encapsulated as Building Blocks (BB) and can be accessed and they can access other resources only by well defined interfaces.

Rodos was implemented as a software framework in C++ with an object oriented application interface (API). it is organized in layers: The lowest layer (1) is responsible for control of the embedded system hardware (HAL:

Hardware abstraction layer). The next layer (2) kernel: administrates the local resources, threads and time. On top of the kernel we have the middleware (layer 3) which enables communication between BBs using a publisher subscriber multicast protocol. And on the top of the middleware the user may implement his applications (layer 4) as a distributed software network of simple BBs. The Building Blocks API on the top of the middleware is a service oriented interface. BBs interact by providing services to other BBs and using services from other BBs.

As mentioned before, the original purpose of Rodos was to control satellites. It was designed as the brain of the Avionic system and introduces for the first time (2001) the NetworkCentric concept. A networkCentric core avionics machine consists of several harmonized components which work together to implement dependable computing in a simple way. In an NetworkCentric system we have a software network of BBs and a hardware Network interconnecting vehicles (radio communication), computers inside of vehicles (buses and point to point links), intelligent devices (attached to buses) and simple devices attached to front end computers. To communicate with (node) external units, including devices and other computing units, each node provides a gateway to the network and around the network's several devices (IO Devs and computing nodes) may be attached to the system. The messages exchange provided by the middleware and gateways is asynchronous, using the publisher-subscriber protocol. No fixed communication paths are established and the system can be reconfigured easily at run-time. For instance, several replicas of the same software can run in different nodes and publish the result using the same topic, without knowing each other. A voter may subscribe to that topic and vote on the correct result. Application can migrate from node to node or even to other vehicles without having to reconfigure the communication system. The core of the middleware distributes messages only locally, but using the integrated gateways to the NetworkCentric network, messages can reach any node and application in the network. The communication in the whole system includes software applications, computing nodes and even IO devices. Publishers make messages public under a given topic. Subscribers (zero, one or more) to a given topic get all messages which are published under this topic. As mentioned before, for this communication there is no difference in which node (computing unit or device) the publisher and subscribers are running and beyond this, they may be any combination of software tasks and hardware devices! To establish a transfer path, both the publisher and the subscriber must share the same topic. A Topic is a pair consisting of a data-type and an integer representing a topic identifier. Both the software middleware and the hardware network switch (called middleware switch), interpret the same publisher/subscriber protocol.[1]

Rodos enables the user to write realtime applications for different architectures in an easy, efficient way. During the development special attention was paid to implement the various features of Rodos in a simple, nevertheless robust way. Unnecessary complexity was avoided to provide the user with a straightforward, clearly arranged system. Rodos supports typical features of realtime operatingsystems, like threads and semaphores.



Rodos Topics for Software and Hardware

Among other features Rodos offers:[2]

- object-oriented C++ interfaces,
- ultra fast booting
- real time priority controlled preemptive multithreading,
- time management (as a central point),
- thread safe communication and synchronisation,

- event propagation

Examples

Hello World

The common Hello world example program looks like this in Rodos.

```
#include "rodos.h"

class HelloWorld : public Thread {
  void run(){
    PRINTF("Hello World!\n");
  }
} helloworld;
```

The class Thread is extended by a custom run() procedure, which writes Hello World to the standard output with PRINTF. All Rodos components needed for application development are accessible via the rodos.h header file.

Threads

Rodos uses *fair priority controlled preemptive scheduling*. The thread with the highest priority is executed while running threads with a lower priority are paused (preemptive multitasking). If there are more than one threads with the same priority, each of them gets a fixed share of computing time and they are executed in turns.

Example:

```
class HighPriorityThread: public Thread{
public:
  HighPriorityThread() : Thread("HiPriority", 25) {
  }
  void run() {
    while(1) {
      xprintf("*");
      suspendCallerUntil(NOW() + 1*SECONDS);
    }
  }
} highprio;

class LowPriorityThread: public Thread {
public:
  LowPriorityThread() : Thread("LowPriority", 10) {
  }

  void run() {
    while(1) {
         xprintf(".");
    }
  }
} lowprio;
```

The thread *LowPriorityThread* constantly writes the character "." and is interrupted every second by the thread *HighPriorityThread*, which writes the character "*".

Topics

Rodos uses so-called *Topics* to enable communication between threads and over gateways between different systems. A *Topic* represents a message of a certain kind. A thread can publish *Topics* as well as subscribe to a *Topic* to receive all messages that belong to a type of message. The message system conforms to the publish–subscribe pattern.

Here is a simple example with one publisher and one subscriber, which both use the *Topic counter1* containing only one integer value.

Example:

```
Topic<long>    counter1(-1, "counter1");

class MyPublisher : public Thread {
public:
    MyPublisher() : Thread("SenderSimple") { }

    void run () {
        long cnt = 0;
        TIME_LOOP(3*SECONDS, 3*SECONDS) {
            PRINTF("Publish: %ld\n", ++cnt);
            counter1.publish(cnt);
        }
    }
} publisher;

class MySubscriber : public SubscriberReceiver<long> {
public:
    MySubscriber() : SubscriberReceiver<long>(counter1) { }
    void put(long &data) {
        PRINTF("Received: %ld\n", data);
    }
}subscriber;
```

The *Publisher-Thread* post every three seconds an ascending counter value, while the *Subscriber-Thread* simply displays the received integer value.

Supported Architectures

Supported instruction set architectures:

- ARM7 (e.g. ARM Cortex-M3, Raspberry Pi's SoC)
- Atmel AVR32
- STM32 32-bit
- PowerPC (PowerPC 405)

Furthermore, Rodos can run as a guest on a different host operating system.

- Linux
- FreeRTOS
- RTEMS
- Windows
- TinyOS
- POSIX-Compatible operating systems

References

1. http://www.montenegros.de/sergio/public/iaa09-coreavionics.pdf
2. http://www.montenegros.de/sergio/public/dasia2009-rodos.pdf

External links

- University of Wuerzburg - The Rodos Framework (http://www8.informatik.uni-wuerzburg.de/wissenschaftforschung/rodos/)