WP3 - Market Analysis

Deliverable D3.2 - Functionalities not Available in Open RTOS

WP3 - Market Analysis: Deliverable D3.2 - Functionalities not Available in Open RTOS
by Adrian Matellanes and Ismael Ripoll

Published October 2002

Table of Contents

List of Tables

Document presentation

Table 1. Project Co-ordinator

Organisation:	UPVLC
Responsible person:	Alfons Crespo
Address:	Camino Vera, 14. CP: 46022, Valencia, Spain
Phone:	+34 9877576
Fax:	+34 9877579
E-mail:	alfons@disca.upv.es

Table 2. Participant List

Role	Id.	Name	Acronym	Country
CO	1	Universidad Politécnica de Valencia	UPVLC	E
CR	2	Scuola Superiore S. Anna	SSSA	I
CR	3	Czech Technical University in Prague	CTU	CZ
CR	4	CEA	CEA	FR
CR	5	UNICONTROLS	UC	CZ
CR	6	MNIS	MNIS	FR
CR	7	VISUAL TOOLS S.A.	VT	E

Table 3. Document Version

Release	Date	Reason of change
1.0	November,	First Release

Chapter 1. Introduction

In deliverable D.1.1, we analyzed several, currently available, Real-Time Operating Systems (RTOSes). We refer to that document for explanations on each one. In this document we present in a briefly manner the functionalities found in those RTOSes and not in Open Source RTOSes.

Since OCERA will develop components for an Open RTOS, we need to consider which functionalities are not available in an Open Source manner and to study the convenience and feasibility of developing such features within OCERA.

The information presented in this document has been considered in the Specification of the OCERA components.

Chapter 2. RTAI

Features found in RT-Linux and missing in RTAI

- EDF dynamic priority scheduling policy. Sporadic scheduler.
- POSIX 1003.13 compatible (mostly).

Features found in RTAI and missing in RT_Linux

- Provide support for MIPS and m68k-nommu processors.
- Priority round-robin scheduling: `SHCED_RR`.
- Support for dynamic memory allocation.
- Interprocess communications:
  - Mailboxes.
  - Four different message facilities: (1) POSIX 1003.b queues; (2) Small (4 bytes) synchronous messages; (3) Extended intertask messaging; and (4) Remote Procedure Calls.
  - A remote execution mechanism called net-rcp, used to execute the system calls in a remote processor.
  - Recursive mutex and error check mutex.
- The priority inheritance used to prevent priority inversion problems.
- Network available as a separate module: RTNet. This module provides the following protocols: IP, ARP, UDP and ICMP.
- The Linux Trace Toolkit (LTT) can be used to debug graphically the applications. Although LTT is not compatible with the POSIX trace standard it work, and is used in a similar way. LTT also provides a graphic tool to visualize the logged events.
- User space hard realtime support. RTAI provides several methods to use the RTAI API from "normal Linux processes", these mechanisms are called: **LXRT**.
- Flexible watchdog utility to monitor time overrun of tasks.
- MatLab & Simulink interface: generate code for RTAI from Simulink Workshop.

Chapter 3. RTEMS

Features of RT-Linux not found in RTEMS

- RT-Linux enables loading and unloading of modules during system run.

  RTEMS is in fact static library of kernel functions which is statically linked with application code and creates monolithic runtime binary image.

- RTEMS has no underlying OS for background tasks
- RTEMS is not designed for standard SMP system usage.

  Task can run on more CPUs (even with different architecture), but distribution of tasks over CPUs is static and defined at compile time.

- RTEMS is more oriented to real embedded system without need of reconfiguration and loading new tasks.

Features found in RTEMS and missing in RT-Linux

- RTEMS has almost complete support for many APIs
  - POSIX 1003.1b API including threads function completeness

    Table 3-1. RTEMS POSIX 1003.1b API

Total number	362
Implemented	301
Unimplemented	21
Unimplementable	16
Partial	2
Dummy	19
Untested	1

  - RTEID/ORKID based Classic API
  - ITRON 3.0 API in progress
  - ADA language version

- Support for more architectures: MC68xxx, MC683xx, Coldfire SH, i386, i960, MIPS, PowerPC, SPARC, AMD A29K, HP PA-RISC, and many BSP targets
- Full dynamic memory management based on memory region manager
- More synchronization and IPC communication primitives not only semaphores and mutexes.

  RTEMS supports semaphores, message queues, events and signals,conditional variables. All with timeout support (forever, time interval, immediate). Binary semaphores with priority inheritance or priority ceiling options.

- User defined timers and timeserver task (thread context for timed callback routines)

- RTEMS enables to use C library (newlib) for applications.

  It includes printf, malloc, open, read/write, fopen, string and memory manipulation functions, date/time, mathematical functions etc.

- Port of FreeBSD TCP/IP stack UDP, TCP , ICMP, DHCP, RARP, TFTP, RPC, FTPD, HTTPD, CORBA
- Filesystem support in progress working IMFS (RAMdisk), FAT and TFTP
- Thread aware debugging over Ethernet or serial port

Chapter 4. QNX

QNX is a full featured RTOS, which implements most of the POSIX realtime extensions.

Some of the QNX functionality is not available in RTLinux but in Linux. QNX is a complete RTOS while RTLinux is just the low level layer of an RTOS system. Some of the QNX functionality is provided at the Linux level (like UNIX processes, filesystems, etc.), other functionality is available in RTLinux. This summary focus only in RTLinux vs. QNX.

Features found in RT-Linux and missing in QNX

- The RTLinux range of user priorities (100000) is wider than the one of QNX.
- RTLinux provides special API functions to implement periodic threads.
- Priority inversion avoidance through immediate priority ceiling.
- EDF dynamic scheduling algorithm: `SCHED_EDF`

Features found in QNX and missing in RT_Linux

- It supports more architectures that RTLinux: x86, ARM, MIPS, PowerPC and SH-4. And multiprocessor support for those architectures that are prepared to: x86 and PowerPC.
- QNX provides both paradigms of concurrency: Normal UNIX processes (weight processes) and threads (lightweight processes), while RTLinux only provides threads.
- It implements the four POSIX scheduling policies:
  - `SCHED_FIFO`
  - `SCHED_RR`
  - `SCHED_OTHER`
  - `SCHED_SPORADIC`
- Processes are executed in protected address space.
- Dynamic memory management: `malloc()`, and `free()` functions.
- The communication and synchronization primitives provided by QNX and not available in RTLinux are: POSIX barriers, read/write locks, message queues and PIPES.
- Priority inversion avoidance through priority inheritance.
- POSIX timers support.
- TCP/IP support. The native networking protocol is Qnet™.
- Wide filesystem compatibility: RAMFS, DOS, ISO9660, Flash, NFS, CIFS, Ext2. And the native filesystem QNX4.
- Propietary Graphic support.
- Visual tool to develop user's interfaces: Photon Application Builder (PhAB).
- QNX provides a complete development environment: GCC, GDB and standard "C" library.

Chapter 5. VxWorks

Features found in RT-Linux and missing in VxWorks

- EDF dynamic priority scheduling policy. Sporadic scheduler.
- RTLinux has two operating modes in relation to clock interrupts: One shot mode and periodic mode. By default RTLinux use one shot mode which provides a high resolution and low overhead.

Features found in VxWorks and missing in RT_Linux

- Wide range of supported processors: Motorola 68k/CPU32/ColdFire/PowerPC, Intel x86, Intel ARM/StrongARM, Hitachi SuperH, MIPS.
- Round-robin scheduling.
- Reduced range of priorities: 256 levels.
- Communications and synchronization primitives not available in RTLinux:
  - Binary, counting, and mutual exclusion semaphores.
  - Message queues.
  - UNIX BSD-style signals and realtime POSIX signals.
  - Sockets.
- Priority inheritance used to control priority inversion.
- Dynamic memory allocation: `malloc()` and `free()`.
- Single memory space, and no memory protection between processes and kernel. Memory protection is optional (module VxSMI).
- The network stack based on the 4.4 BSD TCP/IP. Supported protocols: BSD 4.4 TCP/IP IP, UDP, TCP, IGMP, ICMP, ARP RIP 1/2 SLIP, CSLIP, PPP BOOTP, DNS, DHCP, TFTP FTP, RLOGIN, RSH, TELNET.
- Standard BSD sockets and special *zbuf sockets* (the zero-copy buffer).
- Supported filesystems: FAT, NFS, raw, TrueFFS (Flash file system), ISO9660.
- A graphical timing debugger called WindView®.

Chapter 6. LynxOS

Features found in RT-Linux and missing in LynxOS

- x86 Symmetric Multi Processor (SMP) support. Threads can be assigned to a specific processor.
- RTLinux has two operating modes in relation to clock interrupts: One shot mode and periodic mode. By default RTLinux use one shot mode which provides a high resolution and low overhead.

Features found in LynxOS and missing in RT_Linux

- Wide range of processor support: PowerPC (PPC 601, 603, 604), PowerPC G3 (PPC 75x), PowerPC G4 (PPC 7400,7410,74xx) with Altivec Support, PowerPC IBM 405,440, x86(IA-32), MIPS, ARM9 and Xscale.
- Proprietary Lynx scheduling policy: `SCHED_OTHER`. Which is a priority scheduler with configurable quantum at each priority level.
- LynxOS provides memory protection among user processes and also between user processes and the kernel.
- Dynamic memory management is implemented.
- Communications and synchronization primitives not available in RTLinux:
  - Message queues. Synchronous and asynchronous.
  - Mailboxes.
  - Sockets.
  - Signals.
- Several priority inheritance protocols are available to solve the problem to priority inversion. Two POSIX protocls: `PTHREAD_PRIO_INHERIT` and `PTHREAD_PRIO_PROTECT`, and priority tracking support patented by LynuxWorks™.
- POSIX timers. User can program several virtual timers.
- Provide full state of the art TCP/IP derived from FreeBSD 4.2. And serveral network facilities like DHCP, NFS, Samba, NTP, etc.
- ABI compatibility with Linux V2.4 and glibc V2.2.2.
- Filesystems supported: ROMFS, RAMFS, Flash, NFS, ISO9660, Lynx Fast Filesystem.
- Debugging with GDB enhanced to: multi-thread, system and device driver and remote debug.
- The graphic environment is compatible with the X11 and Motif® standards.