Various Freed Multi-Cores RTOS based Linux

Pu Yiqiao[1,2], ZhouQingguo[1,2]*, She Kairui[1,2], Wu Zhangjin[1,2]
1.Distributed & Embedded Sysytem Lab (DSLab),Lanzhou University, China
2. Engineering Research Center of Open Source Software and Real-Time Systems, Ministry of Education, Lanzhou University, China
zhouqg@lzu.edu.cn, {puyq, shekr, falcon}@dslab.lzu.edu.cn

Abstract

Nowadays multi-cores operating system is widely used in real-time operating systems and embedded operating systems. Multi-cores operating system is stand for a set of system which is based on original system and design for some special purpose. It works as another kernel in the system and only response for several critical areas. These systems are smaller, but most of them will take care of process management which is an important and difficult part in operating system education. So it is more suitable for the operating system teaching. In this paper several open source multi-core operating systems will be introduced especially in process management.

Keywords: Multi-core; real-time OS; process mangement

1. Introduction

Multi-cores RTOS is first appear to implement the hard real-time for Linux during the nineties known as dual kernel [1] [2]. Multi-core RTOS adds a thin layer between hardware and the general purpose operating system to take over the interrupts, and has its own scheduler. In this situation the standard Linux kernel is running in a low priority and only can get CPU and the resource only if there are no real-time tasks. In the beginning it looks like a multithread process with its own scheduler, just like RTLinux. But the structure is clearer in the following systems. They add a hardware abstraction layer between hardware and operating system. This approach still use Linux function such as device drivers in some not critical area, so this make the source code of RT cores is much smaller, they only focus on handling the event processing. These systems have already used in some industrial project, it has been proved can satisfy the hard real-time requirement at the same time reserved the Linux resource. The Multi-Cores RTOS is a standard solution in industry projects. Nowadays Most of them is open source project. They release under the GPL, just like Linux, people can get the source code of them. This make it is possible to read, write and modify the source code during the teaching. Meanwhile it is smaller and more purposiveness than Linux, and is a good place to start with in the education.

In the following part of this paper, RTLinux, RTAI, Xenomai and XtratuM/PaRTiKle will be introduced in section 2, 3, 4 and 5 respectively. And section 6 is a conclusion of the main ideas of this paper.

2. RTLinux

RTLinux [3] is a dual kernel OS which has a transparent, modular, and extensible architecture, the first generation of multi-cores RTOS.

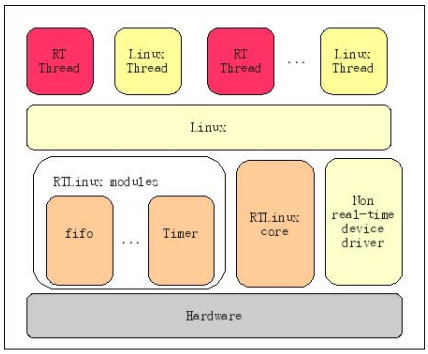

Figure 1. RTLinux architecture

There is a basic rule during the development: “If a service or operation is inherently non real-time, it should be provided in Linux and not in RT components” [4]. This make it just include a small core component and a set of optional components. The core component mainly takes care of interrupt and low level synchronization. For the hard real-time performance

978-1-4244-2511-2/08/$25.00 ©2008 IEEE

RTLinux implement a low latency interrupt handler, this interrupt handler can not be preempted by Linux. And RTLinux also provides a set of function by kernel modules. These modules can be loaded dynamically.

RTLinux can support i386, PPC and ARM architecture.

2.1. Timer and interrupts

Timer and interrupts [5] are two important parts in RTLinux core. RTLinux supports two kinds of hardware timers and provide a set of clock control API which can realize several operations, just like set, read or write time in nanoseconds. The timer module provides a "clock structure" for schedulers and other high level modules. With this operations RTLinux also provide soft timers and theirs management.

The interrupts is divided to global interrupts and local interrupts. The difference between them is where is the interrupt comes from. Local interrupts come from local processors or some special local interrupts controller. But global interrupts use more widely, they come from shared devices.

There are several functions to use the interrupts such as interrupt request, free both in local interrupts and global interrupts. Pay attention that if you are not sure an interrupt is local or global, you'd better treat it as a global interrupt.

There is another kind of interrupt. RTLinux patch change the jump destination of Linux interrupt handler to make RTLinux can takeover the interrupt control after it is loaded in the system. But we still use the original Linux interrupts in non real-time area. This requires the communication with Linux interrupt handlers. If RTLinux receive this kind of interrupt, it will check the status of the system, and the interrupt will be delay until Linux interrupt is enabled. So this kind of interrupt also called soft interrupt.

2.2. Scheduler

The RTLinux scheduler is pure priority driven. The priority is fixed by Rate Monotonic algorithm and the task is scheduled by following these rules:

- Scheduler management the task only by the fixed priority.
- If the cycle of task is less, the priority of the task is higher. Cycle of non-periodic task is infinite.
- High priority task can preempt low priority task.

2.3. Posixio and drivers

Drivers in RTLinux are POSIX compatible. RTLinux has a module rtl_posixio to provide a file system like interface to drivers. This module is like standard Linux register the device in the file system as /dev/filename, and provides open, close, read, write, ioctl and mmap calls as posix definite. Device driver can register the device by following functions and implement their own operation function.

With the rtl_posixio module, RTLinux also provide several inter-process message communication function which is popular in real-time system, fifo and shared memory.

As the earliest Multi-cores RTOS, RTLinux has three versions: RTLinux maintain by FSMLab for commercial, open source project RTLinux/GPL and RTLinuxfree. All three version of RTLinux is widely used in industry and research.

3. RTAI

RTAI [6] [7] is a project which began as "Dipartimento di Ingegneria Aerispaziale del Politecnico di Milano" (DIAPM) in 1996 or 1997. The performance of first version of it is terrible. And another version named NMT-RTL is not very easy to port from v2.0 to v2.2. So a new version in the project appeared, it find out a set of functions which need to modify Linux kernel, and put them into one module called real-time hardware abstraction layer (RTHAL). RTHAL works as a layer between operating system and hardware, and provides interfaces to Linux and RTAI. RTAI stands for Real Time Application

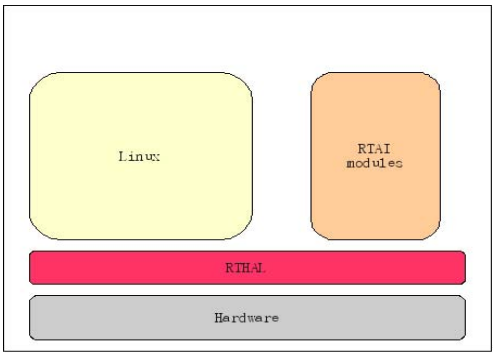

Figure 2. RTAI architecture

Interface. Just like RTLinux, RTAI also implement a group of loadable modules, including RTAI services, LXRT, inter-process communication interface modules (fifo, mailbox, etc.), semaphore, RTAI real-time memory management. At the rest of this part several important modules will be introduced. RTAI can support i386, MIPS, PPC, ARM, m68k-nommu architecture.

3.1. RTHAL

RTHAL is thin layer between hardware and operating system. It performs three main functions:

- Put all the pointers related to Linux kernel into one structure, rthal, to make it easier trapping all the kernel functionalities used in real-time side. This help RTAI takeover at any time when the hard real-time is required.
- Make up the substitutes of above functions and set the pointer of rthal.
- Modify the kernel functions to using the rthal pointers to calls the functions.

By this way, only about 70 lines of code is modified in the kernel. The influence of RTHAL is very slight. And RTAI will build its APIs based on this.

3.2. LXRT

Another character of RTAI is user can use the APIs provided by RTAI to implement a hard or soft real-time application in user space. This set of APIs is called LXRT. LXRT process (running in user space) will call a RTAI process (running in kernel space) functions when it calls a RTAI API function, the parameters will be copied to RTAI peer process when it is executed. To guarantee the real-time performance, the LXRT process also need to get all the memory they use or will used during the process, and keep the memory in physical memory. It can use either Linux system calls or RTAI services without context switching. There are the APIs used to create a LXRT process:

- void * rt_get_adr(unsigned long name);
- unsigned long rt_get_name(void *adr);
- RT_TASK * rt_task_init(int name, int priority, int stack_size, int max_msg_size);
- void rt_make_soft_real_time(void);
- void rt_make_hard_real_time(void);
- void rt_allow_nonroot_hrt(void);
- int rt_register(unsigned long name, void *adr, int type, stuct task_struct *t);
- int rt_drg_on_name(unsigned long name);
- int rt_drg_on_adr(void *adr);

3.3. Scheduler

The scheduler is the most important part of RTAI. Not like RTLinux, the scheduler of RTAI uses a series of mechanisms to meet the real-time request. By using the real-time scheduler, a process can be executed as it is designed. There are three kinds of schedulers in RTAI, UP, SMP and MUP.

- UP is only for uniprocess which is the same as DIAPM-RTL variant scheduler. It can using in one shot timing or periodic timing based on 8254 timer.
- SMP stands for symmetrically multiprocessor, it can use in single oneshot timing or periodic timing based on both 8254 and LOCAL APIC timer. The tasks can be assigned symmetrically on more than one CPU, or be bound to a single CPU. It can be use with UPs when they are based on 8254 timer.
- MUP is only for multiprocessor. The task must be bound to a single CPU which has a LOCAL APIC based timer.

And for complete the scheduler services, there are four set of functions is needed:

- Task functions including real-time task create, delete, and some real-time operations such as set the configuration if the task can be preempt.
- Timing functions is a set of APIs to communication with timers.
- Semaphore functions can used to synchronizing between different thread.
- Inter-task communication functions is used to transfer data between tasks, as the real-time capability required, we can set a dead line in this functions.

3.4. Inter-process communication

RTAI provided plenty of inter-process communication APIs such like FIFO, mailboxes, message queues and remote RPC. These IPCs gives a lot of choice for different application.

- FIFO like UNIX PIPE can be used to exchange data between real-time and non real-time thread. The size of data is fixed.
- Mailbox is ordered like FIFO, but the size of data can be different. Mailbox in RTAI supports multiple senders and receivers and provides sending and receiving functions in several methods: blocking, non-blocking, timed and whole/partial.
- Message queues are divided into four kinds: POSIX message queue, small synchronous message which only has 4 bytes and can be ordered by priority, extended inter-task messaging with any size and Remote Procedure Calls (RPCs) which just add a reply message from receiver than the synchronous message.
- net rpc which allow remote procedure call through network.

After 10 years development, RTAI is used in many real-time control applications, such as robots control and computerized numerical controller.

4. Xenomai

As the requirement of hard real time is increasing, all kinds of real-time operating systems are developed. All these real-time systems has their own APIs, this makes developer spending more and more time to familiar with the APIs. And the developer must rewrite their code when they run their applications in different real-time operating systems. So Xenomai [8] aims to design a framework which supports traditional real-time operating system APIs which makes the existing industrial applications from different real-time operating systems can easily port to a Linux-based environment. And as a real-time operating system Xenomai also keep the hard real-time guarantees. The framework should also be extensible for the future real-time technique. Xenomai implement an abstract real-time nucleus to support the traditional real-time APIs. Xenomai implement the pseudo APIs in different modules, and these modules are called skins. Now Xenomai already has a mount of skins, including real-time APIs, such as VxWorks, pSOS+, VRTX, ulTRON, RTAI and three other skins: POSIX 1003.1b, RTDM and Native skin. Xenomai can runs on ppc,

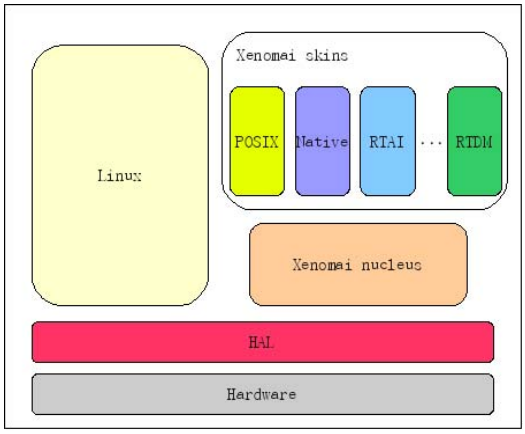

Figure 3. Xenomai architecture

blackfin, arm, x86, x86_64, ia64 and ppc64 platform.

4.1. Xenomai framework

The Xenomai framework wants to emulate traditional real-time operating systems. The first step is just like how to build RTHAL, to find the common behaviors among existed real-time operating systems. Then defined a set of operations which should be put in the nucleus, not all the common behaviors, it has been optimized in terms of efficient. It covers four area: multi-threading, thread synchronization, interrupt management and time management. So the nucleus will provided several group of functions. These functions are considered as a architecture neutral abstraction layer. This layer is not need to be modified when the system is porting.

There are also some Adeos-based [9] real-time abstraction layers in this system for different hardware architecture. This layer is just like the RTHAL in RTAI.

4.2. Xenomai Nucleus

As mentioned above there are some common functions implement in the nucleus of Xenomai. In this section, the four kinds of functions will be introduced.

- Multi-threading

The Multi-threading including thread object (xnthread) and pod (xnpod) abstractions. As any real-time thread, threads can be initialization, suspended, pending, delay, ready to run and running and all this operations can be set a time bound. The threads also support sending signals to threads and asynchronous service. The priority of thread can be increasing or decreasing as the custom configuration, but the priority inversion is prevented. The FPU support is optional.

- Basic synchronization

The synchronization object in nucleus supports for the priority inheritance protocol which is prevent priority inversion and the time bound can be set.

- Interrupt management

The nucleus provides sufficient hooks for different traditional real-time operating system by mechanism which is as simple as possible. The interrupt management can be nesting safely. The operation should be atomicity and the priority of the interrupt is relay to the underlying hardware.

- Time management

The nucleus makes the switch from periodic jiffies to time-stamp counter transparently and can be configured. Nucleus should give a watchdog facility to up level applications to manage the time bound. The real-time skins also can set themselves timer with the global clock value keeping in nucleus.

Xenomai nucleus also provides dynamic memory allocation support for real-time requirements.

4.3. Skins

The conception of skin is proposed by Xenomai. It is a set of APIs which is a simulation of traditional real-time operating system. It used the interfaces provided by nucleus to implement a set of APIs defined by traditional real-time operating system like RTAI, VxWorks, pSOS+, VRTX, ulTRON. There are three its own skins:

- POSIX 1003.1b skin following the POSIX standard can guarantee ultra-low latencies to applications.
- RTDM stands for real-time driver model. It is a skin independent framework for real-time device drive. Under this skin user can develop their own real-time device drive easily.
- Native Xenomai skin is a group of clean RTAI like APIs.

Now Xenomai is used in real-time robotics area just like modular autonomous service robots, real-time 3D larder sensor, etc.

5. XtratuM/PaRTiKle

XtratuM [10] [11] is a hypervisor or a virtual machine monitor for real-time embedded system. It is developed to make the RTLinux virtualization mechanism better. It is also focus on shared devices between non real-time partition and real-time partition such as clock, timer and interrupts. A character of XtratuM is that it provides divided memory space for the domain running above it to make the system robust. PaRTiKle [12] is a real-time partition running in XtratuM under the POSIX PSE51 interface. PaRTiKle is also can run as a stand-alone system and a regular process in Linux. This is the next generation of RTLinux/GLP. The architecture is showing in figure 4.

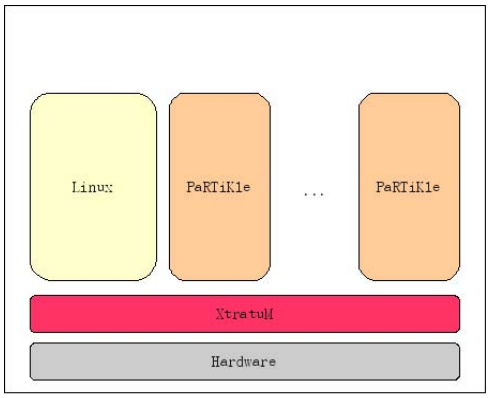

Figure 4. XtratuM/PaRTiKle architecture

XtratuM can support i386 and LEON2 processor. PaRTiKle as a stand-alone system can support i386 and arm processor. In the following part, XtratuM and PaRTiKle will be introduced separated.

5.1. XtratuM

XtratuM is designed for real-time and embedded system. This require the designers to cut virtualizes to the minimal size in the same time keep the real-time capability. So XtratuM is not virtualization of all the hardware architecture, it basically offers some obligatory parts as following:

- Interrupts

This is a continuously part in multi-cores RTOS. XtratuM also taking over the hardware interrupts to control the machine. It replaces the old interrupt handler in Linux when it is started. It will make judgment whether the interrupt is coming from which partition and provide the hooks to the partitions. The operating system runnings above XtrautM just deal with the virtual interrupts.

- Timer

Because of the timer is relay to hardware architecture and the published version of XtratuM just run in the Intel x86 architecture. XtratuM just provide two virtual timers: PIT and APIC timer, and a group of high-level API to work with the virtual timers.

- Virtual Memory

XtratuM creates different memory map for each operating system. Obviously, the memory space of partitions is isolated. This is good to make the partitions robust. If one partition is crashed, except the root domain, the other partitions will not be influence. But there are some lacks of features just like the lack of IPCs. The solutions will be mentioned in the following section.

5.2 PaRTiKle

PaRTiKle is an embedded real-time operating system. The design is trying to improve the portability, customization and the maintainability of the system. The system can be executed under different environment. So far we can run PaRTiKle above a bared machine, as a Linux process and a domain of XtratuM. The system also supports several programming language: ADA, C, C++ and JAVA.

PaRTiKle also has its own kernel space and user space which interact by the system calls in PaRTiKle. In the kernel of PaRTiKle there is a hardware abstraction layer which makes PaRTiKle can run in different environments provides a set of hardware independent functions including:

- Scheduler in PaRTiKle is a simple rate-monotonic scheduling scheduler.
- Physical memory management to management the physical memory.
- Timer and clock management functions which can provide several different virtual timers.
- Kernel library is a minimal C library for PaRTiKle kernel.
- System calls like the Linux system calls is used to provide kernel services for user space applications

As the kernel library can not be used by user space applications, PaRTiKle implements several libraries in user space including ADA, C, C++ and Java. The C library is installed by default and others is optional. So the developer can write their programs just like under Linux, the code is same but compile with different library.

5.3 IPCs

As mentioned above, the IPCs are lack in XtratuM/PaRTiKle at the beginning. But there are two kinds of IPCs you can choice from the system: fifo and shared memory.

- XM-FIFO is a first in first out data queue with fixed size. It is non-blocking and lock-free.
- XM-SHM is a portion of memory can be accessed by multi process.

XtratuM/PaRTiKle is a new approach so it is only used in research area.

6. Conclusions

This article gives a general introduction of Multi-cores RTOS and four popular real-time operating systems which has the architecture. From the introduction we can see the track of the development of multi-cores RTOS. The architecture of the system is more and more clear definiteness and the modularity of the system is higher. By the clean API documents, it is easy to implement experiment with it. And all of these systems are open source project which can easily get from internet and support by seasoned open source community. Both traditional and new technique in real-time operating system can be find in these developed systems will also helpful to the real-time operating system education.

7. References

[1] SM Bicer, F Pilhofer, G Bardouleau, J Smith, “Next-generation hard real-time on POSIX-based Linux”, *embedded Control Europe*, 06/2006.

[2] R Lehrbaum, “Using Linux in Embedded and Real-Time Systems”, *Linux Journal*, 2000

.[3] Victor YODAIKEN, Cort DOUGAN,Michael BARANOV, “RTLinux/RTCore dual kernel real-time operating system”, FSMLabs Inc.

[4] V Yodaiken, M Barabanov, “RTLinux Version two”, *Real Time Linux Workshop*, 1999

[5] She Kairui, Bai Shuwei, Zhou Qingguo, Nicholas Mc Guire, Li Lian, “Analyzing RTLinux/GPL Source Code for Education”, *8th Real Time Linux Workshop*, 2006

[6] Pierre Cloutier, Paolo Montegazza, Steve Papachar-alambous, Ian Soanes, Stuart Hughes, Karim aghmour, “DIAPM-RTAI position paper”, *2nd Real Time Linux Workshop*, Orlando, FL, November 2000

[7] D. Beal, E. Bianchi, L. Dozio, S. Hughes, P. Mantegazza, S. Papacharalambous, “RTAI: Real-Time Application Interface”, *Linux Journal*, April 2000.

[8] P. Gerum, “Xenomai - implementing a rtos emulation framework on Gnu/linux”, http://www.xenomai.org, 2004

[9] Karim Yaghmour, “Building a Real-Time Operating System on Top of the Adaptive Domain Environment for Operating Systems”, Opersys Inc., Feb 2001

[10] Miguel Masmano, Ismael Ripoll, Alfons Crespo, “XtratuM Overview”, http://www.xtratum.org/papers

/Intro/xtratum_intro.html

[11] M. Masmano, I. Ripoll, and C. Crespo, “An overview of the XtratuM nanokernel,” *the Workshop on Operating System Platforms for Embedded Real-Time Applications*, Palma de Mallorca, Spain, July 2005.

[12] S. Peiro, M. Masmano, I. Ripoll, and A. Crespo, “PaRTiKle OS, a replacement for the core of RTLinux-GPL”, *9th real time linux workshop,* Austria, Nov 2007