Module 6

Embedded System Software

Lesson 32

Commercial Real-Time Operating Systems

Specific Instructional Objectives

At the end of this lesson, the student would be able to:

- Get an understanding of open software
- Know the historical background under which POSIX was developed
- Get an overview of POSIX
- Understand the Real-Time POSIX standard
- Get an insight into the features of some of the popular Real-Time OS: PSOS, VRTX, VxWorks, QNX, μC/OS-II, RT-Linux, Lynx, Windows CE

1. Introduction

Many real-time operating systems are at present available commercially. In this lesson, we analyze some of the popular real-time operating systems and investigate why these popular systems cannot be used across all applications. We also examine the POSIX standards for RTOS and their implications.

1.1. POSIX

POSIX stands for Portable Operating System Interface. "X" has been suffixed to the abbreviation to make it sound Unix-like. Over the last decade, POSIX has become an important standard in the operating systems area including real-time operating systems. The importance of POSIX can be gauzed from the fact that nowadays it has become uncommon to come across a commercial operating system that is not POSIX-compliant. POSIX started as an open software initiative. Since POSIX has now become overwhelmingly popular, we discuss the POSIX requirements on real-time operating systems. We start with a brief introduction to open software movement and then trace the historical events that have led to the emergence of POSIX. Subsequently, we highlight the important requirements of real-time POSIX.

1.2. Open Software

An *open system* is a vendor neutral environment, which allows users to intermix hardware, software, and networking solutions from different vendors. Open systems are based on open standards and are not copyrighted, saving users from expensive intellectual property right (IPR) law suits. The most important characteristics of open systems are: interoperability and portability. Interoperability means systems from multiple vendors can exchange information among each other. A system is portable if it can be moved from one environment to another without modifications. As part of the open system initiative, open software movement has become popular.

Advantages of open software include the following: It reduces cost of development and time to market a product. It helps increase the availability of add-on software packages. It enhances the ease of programming. It facilitates easy integration of separately developed modules. POSIX is an off-shoot of the open software movement.

Open Software standards can be divided into three categories:

Open Source: Provides portability at the source code level. To run an application on a new platform would require only compilation and linking. ANSI and POSIX are important open source standards.

Open Object: This standard provides portability of unlinked object modules across different platforms. To run an application in a new environment, relinking of the object modules would be required.

Open Binary: This standard provides complete software portability across hardware platforms based on a common binary language structure. An open binary product can be portable at the executable code level. At the moment, no open binary standards.

The main goal of POSIX is application portability at the source code level. Before we discuss about RT-POSIX, let us explore the historical background under which POSIX was developed.

1.3. Genesis of POSIX

Before we discuss the different features of the POSIX standard in the next subsection, let us understand the historical developments that led to the development of POSIX.

Unix was originally developed by AT&T Bell Labs. Since AT&T was primarily a telecommunication company, it felt that Unix was not commercially important for it. Therefore, it distributed Unix source code free of cost to several universities. UCB (University of California at Berkeley) was one of the earliest recipient of Unix source code.

AT&T later got interested in computers, realized the potential of Unix and started developing Unix further and came up with Unix V. Meanwhile, UCB had incorporated TCP/IP into Unix through a large DARPA (Defense Advanced Research Project Agency of USA) project and had come up with BSD 4.3 and C Shell. With this, the commercial importance of Unix started to grow rapidly. As a result, many vendors implemented and extended Unix services in different ways: IBM with its AIX, HP with its HP-UX, Sun with its Solaris, Digital with its Ultrix, and SCO with SCO-Unix. Since there were so many variants of Unix, portability of applications across Unix platforms became a problem. It resulted in a situation where a program written on one Unix platform would not run on another platform.

The need for a standard Unix was recognized by all. The first effort towards standardization of Unix was taken by AT&T in the form of its SVID (System V Interface Definition). However, BSD and other vendors ignored this initiative. The next initiative was taken under ANSI/IEEE, which yielded POSIX.

1.4. Overview of POSIX

POSIX is an off-shoot of the open software movement, and portability of applications across different variants of Unix operating systems was the major concern. POSIX standard defines only interfaces to operating system services and the semantics of these services, but does not specify how exactly the services are to be implemented. For example, the standard does not specify whether an operating system kernel must be single threaded or multithreaded or at what priority level the kernel services are to be executed.

The POSIX standard has several parts. The important parts of POSIX and the aspects that they deal with, are the following:

Open Source: Provides portability at the source code level. To run an application on a new platform would require only compilation and linking. ANSI and POSIX are important open source standards.

- **POSIX 1 :** system interfaces and system call parameters
- **POSIX 2 :** shells and utilities
- **POSIX 3 :** test methods for verifying conformance to POSIX
- **POSIX 4 :** real-time extensions

1.5. Real-Time POSIX Standard

POSIX.4 deals with real-time extensions to POSIX and is also known as POSIX-RT. For an operating system to be POSIX-RT compliant, it must meet the different requirements specified in the POSIX-RT standard. The main requirements of the POSIX-RT are the following:

- **Execution scheduling:** An operating system to be POSIX-RT compliant must provide support for real-time (static) priorities.
- **Performance requirements on system calls:** It specifies the worst case execution times required for most real-time operating services.
- **Priority levels:** The number of priority levels supported should be at least 32.
- **Timers:** Periodic and one shot timers (also called watch dog timer) should be supported. The system clock is called CLOCK REALTIME when the system supports real-time POSIX.
- **Real-time files:** Real-time file system should be supported. A real-time file system can pre-allocate storage for files and should be able to store file blocks contiguously on the disk. This enables to have predictable delay in file access in virtual memory system.
- **Memory locking:** Memory locking should be supported. POSIX-RT defines the operating system services: mlockall() to lock all pages of a process, mlock() to lock a range of pages, and mlockpage() to lock only the current page. The unlock services are munlockall(), munlock(), and munlockpage. Memory locking services have been introduced to support deterministic memory access.
- **Multithreading support:** Real-time threading support is mandated. Real-time threads are schedulable entities of a real-time application that have individual timeliness constraints and may have collective timeliness constraints when belonging to a runnable set of threads.

1.6. A Survey of Contemporary Real-Time Operating Systems

In this section, we briefly survey the important feature of some of the popular real-time operating systems that are being used in commercial applications.

1.6.1. PSOS

PSOS is a popular real-time operating system that is being primarily used in embedded applications. It is available from Wind River Systems, a large player in the real-time operating system arena. It is a host-target type of real- time operating system. PSOS is being used in

several commercial embedded products. An example application of PSOS is in the base stations of the cellular systems.

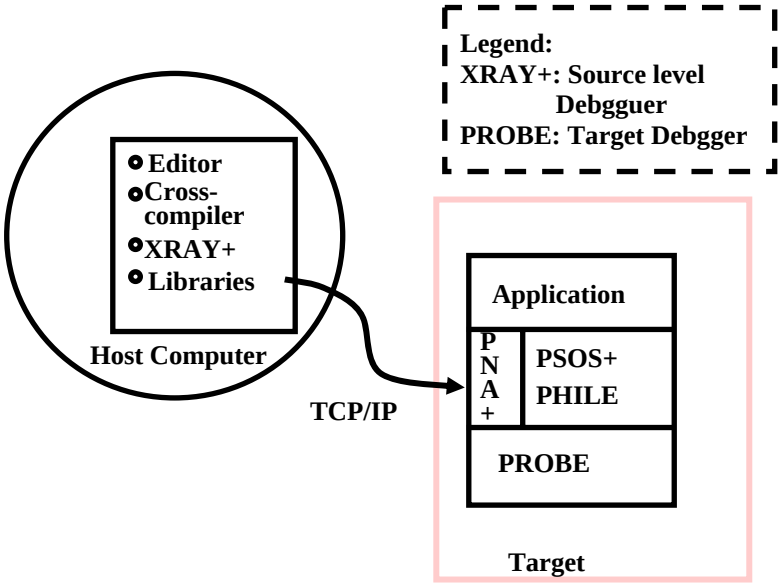

Fig. 32.1 PSOS-based Development of Embedded Software

PSOS-based application development has schematically been shown in Fig. 32.1. The host computer is typically a desktop. Both Unix and Windows hosts are supported. The target board contains the embedded processor, ROM, RAM, etc. The host computer runs the editor, cross-compiler, source-level debugger, and library routines. On the target board PSOS+, and other optional modules such as PNA+, PHILE, and PROBE are installed on a RAM. PNA+ is the network manager. It provides TCP/IP communication over Ethernet and FDDI. It conforms to Unix 4.3 (BSD) socket syntax and is compatible with other TCP/IP-based networking standards such as ftp and NFS. Using these, PNA+ provides efficient downloading and debugging communication between the target and the host. PROBE+ is the target debugger and XRAY+ is the source-level debugger. The application development is done on the host machine and is downloaded to the target board. The application is debugged using the source debugger (XRAY+). Once the application runs satisfactorily, it is fused on a ROM and installed on the target board.

We now highlight some important features of PSOS. PSOS consists of 32 priority levels. In the minimal configuration, the foot print of the operating system is only 12KBytes. For sharing critical resources among real-time tasks, it supports priority inheritance and priority ceiling protocols. It support segmented memory management. It allocates tasks to memory regions. A memory region is a physically contiguous block of memory. A memory region is created by the operating system in response to a call from an application.

In most modern operating systems, the control jumps to the kernel when an interrupt occurs. PSOS takes a different approach. The device drivers are outside the kernel and can be loaded and removed at the run time. When an interrupt occurs, the processor jumps directly to the ISR (interrupt service routine) pointed to by the vector table.

The intention is not only to gain speed, but also to give the application developer complete control over interrupt handling.

1.6.2. VRTX

VRTX is a POSIX-RT compliant operating system from Mentor Graphics. VRTX has been certified by the US FAA (Federal Aviation Agency) for use in mission and life critical applications such as avionics. VRTX has two multitasking kernels: VRTXsa and VRTXmc.

VRTXsa is used for large and medium applications. It supports virtual memory. It has a POSIX-compliant library and supports priority inheritance. Its system calls are deterministic and fully preemptable. VRTXmc is optimized for power consumption and ROM and RAM sizes. It has therefore a very small foot print. The kernel typically requires only 4 to 8 Kbytes of ROM and 1KBytes of RAM. It does not support virtual memory. This version is targeted for cell phones and other small hand-held devices.

1.6.3. VxWorks

VxWorks is a product from Wind River Systems. It is host-target system. The host can be either a Windows or a Unix machine. It supports most POSIX-RT functionalities. VxWorks comes with an integrated development environment (IDE) called Tornado. In addition to the standard support for program development tools such as editor, cross-compiler, cross-debugger, etc. Tornado contains VxSim and WindView. VxSim simulates a VxWorks target for use as a prototyping and testing environment. WindView provides debugging tools for the simulator environment. VxMP is the multiprocessor version of VxWorks.

VxWorks was deployed in the Mars Pathfinder which was sent to Mars in 1997. Pathfinder landed in Mars, responded to ground commands, and started to send science and engineering data. However, there was a hitch: it repeatedly reset itself. Remotely using trace generation, logging, and debugging tools of VxWorks, it was found that the cause was unbounded priority inversion. The unbounded priority inversion caused real-time tasks to miss their deadlines, and as a result, the exception handler reset the system each time. Although VxWorks supports priority inheritance, using the remote debugging tool, it was found to have been disabled in the configuration file. The problem was fixed by enabling it.

1.6.4. QNX

QNX is a product from QNX Software System Ltd. QNX Neutrino offers POSIX-compliant APIs and is implemented using microkernel architecture.

The microkernel architecture of QNX is shown in Fig. 32.2. Because of the fine grained scalability of the micro- kernel architecture, it can be configured to a very small size – a critical advantage in high volume devices, where even a 1% reduction in memory costs can return millions of dollars in profit.

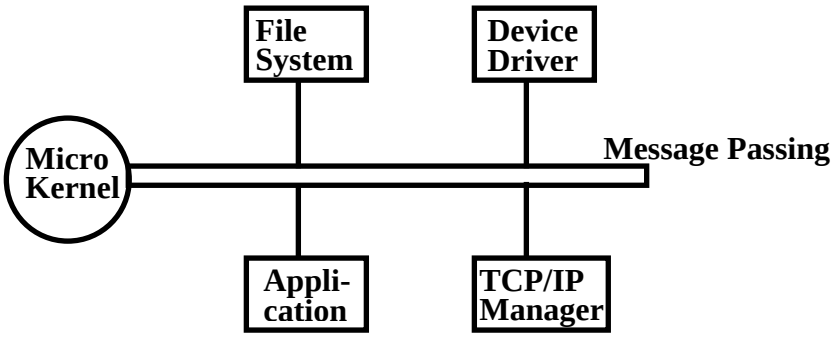

Fig. 32.2 Microkernel Architecture of QNX

1.6.5. μC/OS-II

μC/OS-II is a free RTOS, easily available on Internet. It is written in ANSI C and contains small portion of assembly code. The assembly language portion has been kept to a minimum to make it easy to port it to different processors. To date, μC/OS-II has been ported to over 100 different processor architectures ranging from 8-bit to 64-bit microprocessors, microcontrollers, and DSPs. Some important features of μC/OS-II are highlighted in the following.

- μC/OS-II was designed so that the programmer can use just a few of the offered services or select the entire range of services. This allows the programmer to minimize the amount of memory needed by μC/OS-II on a per-product basis.
- μC/OS-II has a fully preemptive kernel. This means that μC/OS-II always ensures that the highest priority task that is ready would be taken up for execution.
- μC/OS-II allows up to 64 tasks to be created. Each task operates at a unique priority level. There are 64 priority levels. This means that round-robin scheduling is not supported. The priority levels are used as the PID (Process Identifier) for the tasks.
- μC/OS-II uses a partitioned memory management. Each memory partition consists of several fixed sized blocks. A task obtains memory blocks from the memory partition and the task must create a memory partition before it can be used. Allocation and deallocation of fixed-sized memory blocks is done in constant time and is deterministic. A task can create and use multiple memory partitions, so that it can use memory blocks of different sizes.
- μC/OS-II has been certified by Federal Aviation Administration (FAA) for use in commercial aircraft by meeting the demanding requirements of its standard for software used in avionics. To meet the requirements of this standard it was demonstrated through documentation and testing that it is robust and safe.

1.6.6. RT Linux

Linux is by large a free operating system. It is robust, feature rich, and efficient. Several real-time implementations of Linux (RT-Linux) are available. It is a self-host operating system (see Fig. 32.3). RT-Linux runs along with a Linux system. The real-time kernel sits between the hardware and the Linux system. The RT kernel intercepts all interrupts generated by the hardware. Fig. 32.12 schematically shows this aspect. If an interrupt is to cause a real-time task

to run, the real-time kernel preempts Linux, if Linux is running at that time, and lets the real-time task run. Thus, in effect Linux runs as a task of RT-Linux.

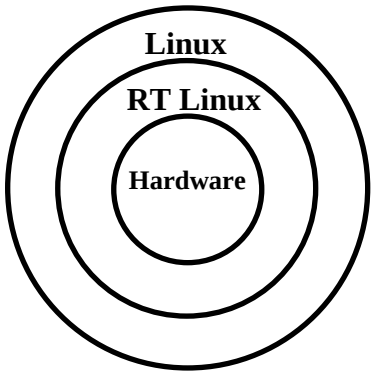

Fig. 32.3 Structure of RT Linux

The real-time applications are written as loadable kernel modules. In essence, real-time applications run in the kernel space.

In the approach taken by RT Linux, there are effectively two independent kernels: real-time kernel and Linux kernel. Therefore, this approach is also known as the *dual kernel approach* as the real-time kernel is implemented outside the Linux kernel. Any task that requires deterministic scheduling is run as a real-time task. These tasks preempt Linux whenever they need to execute and yield the CPU to Linux only when no real-time task is ready to run.

Compared to the microkernel approach, the following are the shortcomings of the dual-kernel approach.

- **Duplicated Coding Efforts:** Tasks running in the real-time kernel can not make full use of the Linux system services – file systems, networking, and so on. In fact, if a real-time task invokes a Linux service, it will be subject to the same preemption problems that prohibit Linux processes from behaving deterministically. As a result, new drivers and system services must be created specifically for the real-time kernel – even when equivalent services already exist for Linux.
- **Fragile Execution Environment:** Tasks running in the real-time kernel do not benefit from the MMU-protected environment that Linux provides to the regular non-real-time processes. Instead, they run unprotected in the kernel space. Consequently, any real-time task that contains a coding error such as a corrupt C pointer can easily cause a fatal kernel fault. This is serious problem since many embedded applications are safety-critical in nature.
- **Limited Portability:** In the dual kernel approach, the real-time tasks are not Linux processes at all; but programs written using a small subset of POSIX APIs. To aggravate the matter, different implementations of dual kernels use different APIs. As a result, real-time programs written using one vendor's RT-Linux version may not run on another's.
- **Programming Difficulty:** RT-Linux kernels support only a limited subset of POSIX APIs. Therefore, application development takes more effort and time.

1.6.7. Lynx

Lynx is a self host system. The currently available version of Lynx (Lynx 3.0) is a microkernel-based real-time operating system, though the earlier versions were based on monolithic design. Lynx is fully compatible with Linux. With Lynx's binary compatibility, a Linux program's binary image can be run directly on Lynx. On the other hand, for other Linux compatible operating systems such as QNX, Linux applications need to be recompiled in order to run on them. The Lynx microkernel is 28KBytes in size and provides the essential services in scheduling, interrupt dispatch, and synchronization. The other services are provided as kernel plug-ins (KPIs). By adding KPIs to the microkernel, the system can be configured to support I/O, file systems, sockets, and so on. With full configuration, it can function as a multipurpose Unix machine on which both hard and soft real-time tasks can run. Unlike many embedded real-time operating systems, Lynx supports memory protection.

1.6.8. Windows CE

Windows CE is a stripped down version of Windows, and has a minimum footprint of 400KBytes only. It provides 256 priority levels. To optimize performance, all threads are run in the kernel mode. The timer accuracy is 1 msec for sleep and wait related APIs. The different functionalities of the kernel are broken down into small non-preemptive sections. So, during system call preemption is turned off for only short periods of time. Also, interrupt servicing is preemptable. That is, it supports nested interrupts. It uses memory management unit (MMU) for virtual memory management.

Windows CE uses a priority inheritance scheme to avoid priority inversion problem present in Windows NT. Normally, the kernel thread handling the page fault (i.e. DPC) runs at priority level higher than NORMAL (refer Sec. 4.5.2). When a thread with priority level NORMAL suffers a page fault, the priority of the corresponding kernel thread handling this page fault is raised to the priority of the thread causing the page fault. This ensures that a thread is not blocked by another lower priority thread even when it suffers a page fault.

1.6.9. Exercises

1. State whether the following statements are True or False. Justify your answer in each case.
   a. In real-time Linux (RT-Linux), real-time processes are scheduled at priorities higher than the kernel processes.
   b. EDF scheduling of tasks is commonly supported in commercial real-time operating systems such as PSOS and VRTX.
   c. POSIX 1003.4 (real-time standard) requires that real-time processes be scheduled at priorities higher than kernel processes.
   d. POSIX is an attempt by ANSI/IEEE to enable executable files to be portable across different Unix machines.
2. What is the difference between block I/O and character I/O? Give examples of each. Which type of I/O is accorded higher priority by Unix? Why?
3. List four important features that a POSIX 1003.4 (Real-Time standard) compliant operating system must support. Is preemptability of kernel processes required by POSIX 1003.4? Can a Unix-based operating system using the preemption-point technique claim to be POSIX 1003.4 compliant? Explain your answers.

4. Suppose you are the manufacturer of small embedded components used mainly in consumer electronics goods such as automobiles, MP3 players, and computer-based toys. Would you prefer to use PSOS, WinCE, or RT-Linux in your embedded component? Explain the reasons behind your answer.
5. What is the difference between a system call and a function call? What problems, if any, might arise if the system calls are invoked as procedure calls?
6. Explain how a real-time operating system differs from a traditional operating system. Name a few real-time operating systems that are commercially available.
7. What is open software? Does an open software mandate portability of the executable files across different platforms? Name an open software standard for real-time operating systems. What is the advantage of using an open software operating system for real-time application development? What are the pros and cons of using an open software product in program development compared to a proprietary product?
8. Identify at least four important advantages of using VxWorks as the operating system for real-time applications compared to using Unix V.3.
9. What is an open source standard? How is it different from open object and open binary standards? Give some examples of popular open source software products.
10. Can multithreading result in faster response times (compared to single threaded tasks) even in uniprocessor systems? Explain your answer and identify the reasons to support your answer.

References (Lessons 24 - 28)

1. C.M. Krishna and Shin K.G., Real-Time Systems, Tata McGraw-Hill, 1999.
2. Philip A. Laplante, Real-Time System Design and Analysis, Prentice Hall of India, 1996.
3. Jane W.S. Liu, Real-Time Systems, Pearson Press, 2000.
4. Alan C. Shaw, Real-Time Systems and Software, John Wiley and Sons, 2001.
5. C. SivaRam Murthy and G. Manimaran, Resource Management in Real-Time Systems and Networks, MIT Press, 2001.
6. B. Dasarathy, Timing Constraints of Real-Time Systems: Constructs for Expressing Them, Methods for Validating Them, IEEE Transactions on Software Engineering, January 1985, Vol. 11, No. 1, pages 80-86.
7. Lui Sha, Ragunathan Rajkumar, John P. Lehoczky, Priority inheritance protocols: An approach to real-time synchronization,, IEEE Transactions on Computers, 1990, Vol. 39, pages 1175-1185.

Source:http://www.nptel.ac.in/courses/Webcourse-contents/IIT%20Kharagpur/Embedded%20systems/Pdf/Lesson-32.pdf