

A Survey of Embedded Operating System

Catherine Lingxia Wang, Bo Yao, Yang Yang, Zhengyong Zhu

Abstract

This paper presents a survey of several major embedded operating systems. It analyzes several design issues of embedded operating system, such as architecture, memory management, IPC, process management, network support, and the impact of hardware limitation and application requirement of embedded system.

We analyzed three prevailing embedded operating systems: Windows CE, embedded Linux and QNX. Finally we gave an overview on the considerations why these decisions are made.

Introduction

Embedded system is application-oriented special computer system which is scalable on both software and hardware. It can satisfy the strict requirement of functionality, reliability, cost, volume, and power consumption of the particular application.

With rapid development of IC design and manufacture, CPUs became cheap. Lots of consumer electronics have embedded CPU and thus became embedded systems. For example, PDAs, cellphones, point-of-sale devices, VCRs, industrial robot control, or even your toasters can be embedded system. There is more and more demand on the embedded system market. Some report expects that the demand on embedded CPUs is 10 times as large as general purpose PC CPUs.

As applications of the embedded systems become more complex, the operating system support and development environment became crucial. In this paper, we mainly analyze three major embedded operating systems, QNX 4 RTOS, Windows CE and embedded Linux. Windows CE and embedded Linux are most widely used embedded operation systems. QNX is a relatively simple one and can fit in some simple applications.

QNX 4 RTOS

The QNX 4 RTOS is developed by QNX Software Systems Ltd. for the applications in consumer electronics, telecommunications, automotive systems, medical instrumentation which need high reliability, superior performance, sophisticated functionality, and massive scalability. It's small, scalable, extensible, and fast. A number of design innovations were developed for QNX 4 RTOS to deliver the full performance of the hardware, which also help us to look in the impact of embedded system hardware limitation on the OS design.

Embedded Linux

In the past two years, Linux has become popular on embedded devices—especially consumer gadgets, telecom routers and switches, Internet appliances and automotive applications.

Because of the modular nature of Linux, it is easy to slim down the operating environment by removing utility programs, tools, and other system services that are not needed in an embedded environment. You can strip the standard Linux kernel to the barest as needed. One of the major advantages of Linux is that it is a fully functional OS, with support for network that is becoming a very important requirement in embedded systems because people need to "compute anywhere, anytime". Because you can add or unload modules from the kernel at runtime, this makes embedded Linux very flexible. It is more encouraging that the Linux code is widely available, portable to any processor, scalable and stable. Because it is open, Linux

doesn't require the user to pay license fees or royalties—particularly important to developers of consumer electronics, who have narrow margins.

Windows CE

Windows CE is first introduced in the Handheld PC (H/PC) set of products in November 1996. But later became a highly configurable operating system after Microsoft released the windows CE OEM Adaptation Kit in March 1997. Several advantages for Windows CE includes a subset of the Win32 API that addresses the most commonly needed services, a low overhead device driver model and built-in power management. Windows CE can be ported to a broad range of business, consumer and industrial devices. The application of Windows CE includes consumer electronics like handheld PC, AutoPC, video game player and digital camera, and industrial products like barcode reader and programmable logic controllers.

Comparison

1. Hardware specification

There are various hardware platforms for embedded system. The most popular ones include x86, MIPS, PowerPC, Hitachi SH, PowerPC and Strong Arm processors. On one hand, the embedded operating systems are adapted to run on most of the processor available to embedded system. On the other hand, they also have some own restrictions.

Processor

QNX can run on all generic x86-based processors (386 and up).

Linux is currently available to run on virtually every general-purpose microprocessor and to support the most common processors used in the embedded world, including the ARM, Strong Arm, MIPS, Hitachi SH, Motorola/IBM PowerPC, and x86-compatible families. EHOE vendors, such as Red Hat, also offer porting services that will quickly move a Linux EHOE to the new processor architecture.

Windows CE require the underlying CPU to use a flat 32-bit address space and support kernel- and user-mode operation, to support virtual memory using an MMU and to be little-endian. Fortunately, Now Windows CE can target various platforms like x86, MIPS, Hitachi SH3 and SH4, PowerPC and Strong Arm processors.

Memory

Memory is always a precious resource on embedded system. The embedded operating systems thus make large effort to reduce its memory occupation size. QNX is the smallest OS among the three. It has a very small kernel of about 12k bytes. The kernel can easily fit into the on-chip cache. Windows CE and Linux require more memory because they are more complicated. Windows CE needs 350KB for a minimal system with the kernel and some communication support. Linux needs 125-256 KB for a reasonable configured kernel, and over 100 KB for other components. Both Windows CE and Linux are configurable, so the system size varies for different application.

2. Architecture

Embedded operating system use either microkernel or a modular architecture to make them easily tailored to fit in different application requirement.

QNX contains a very small microkernel surrounded by a team of cooperating processes that provide higher-level OS services. The QNX microkernel implement four services: 1) Interprocess communication. 2) Low-level network communication. 3) Process scheduling. 4) Interrupt dispatching. The OS service processes are optional, and user can choose which one is needed for their applications. This kind of design makes QNX flexible for different kinds of applications which require different kinds of OS services.

The operating system architecture of Windows CE is a hierarchical one. At the bottom lies the device drivers and OAL(OEM Abstraction Layer). They are implemented by the OEM when porting Windows CE. Above them lie the Graphics, Windowing and Events Subsystem (GWES), the kernel itself and the communication stacks. This layer is implemented by Microsoft. The Remote API capability is built on top of the communication functionality. On top of the kernel lies the database and file system. This is accessed by the RAPI calls, and is made available to the applications via the Win32 interface. Application execute in their own address space and interact with the rest of Windows CE via the Win32 system call interface.

Linux is also a layering structure and comprised of modules, such as Linux kernel, file system, device drivers and network protocols. Embedded Linux takes the Linux kernel and extract the necessary modules as needed. Within the kernel layer, Linux is composed of five major subsystems: the process scheduler (sched), the memory manager (mm), the virtual file system (vfs), the network interface (net), and the inter-process communication (ipc). Conceptually, the clustering of the components composes the Linux Kernel and each subsystem is an independent component of the Linux kernel. However, there are still quite some interdependence among the five subsystems in the concrete architecture.

3. Process management

QNX process manager dose not reside in QNX microkernel. Although it shares the same address space as the microkernel (and is the only process to do so), the Process Manager runs as other processes, scheduled by microkernel and uses the Microkernal's message passing primitives to communicates with other processes in the system. Process manager is responses for the creation of processes and management of process resource.

The scheduling of processes is managed by the microkernel scheduler. The microkernel scheduler makes scheduling decision when:

- 1) A process becomes unblocked.
- 2) The timeslice for a running process expires
- 3) A running process is preempted.

Every process is assigned a priority. The scheduler selects the next process to run by looking at the process priority. When more than one process at the same priority are ready to run, the scheduler uses three scheduling methods:

- 1) FIFO scheduling.
- 2) Round-robin scheduling.
- 3) Adaptive scheduling.

The QNX is a fully preemptible system which supports real-time operating system

Windows CE support both processes and threads. Full memory protection applies to application processes. Thread scheduling is carried out preemptively, using 8 different priority levels. Windows CE also use Wait Functions and Wait Objects for synchronization. Windows CE is a preemptive multitasking operating system. It allows multiple applications, or processes to run within the system at the same time. It supports a maximum of 32 simultaneous processes. A process consists of one or more threads.

Windows CE uses a priority-based time-slice algorithm to schedule the execution of threads. It supports 8 discrete priority levels. Preemption is based solely on the thread's priority. Threads with a higher priority are scheduled to run first. Threads at the same priority level run in a round-robin fashion with each thread receiving a slice of execution time. Like other Windows operating system, Windows CE offers a rich set of "wait objects" for thread synchronization. These include critical section, event and mutex object. These

wait object allow a thread to block its own execution until the specified object changes. Windows CE queues mutex, critical section and event requests in "FIFO-by-priority" order: a different FIFO queue is defined for each of the 8 priority levels.

Real-time application use interrupts as a way of ensuring that external events are quickly noticed by the operating system. Windows CE balance performance and ease of implementation by splitting interrupt processing into two steps: an interrupt service routine and an interrupt service thread. Each hardware interrupt request line is associated with one ISR. When an interrupt occurs, the kernel calls the registered ISR for that interrupt. The ISR, the kernel-mode portion of interrupt processing is kept as short as possible. It returns an interrupt ID to the kernel. The kernel sets the associated event according to the ID. The interrupt service thread is waiting on that event. ISTs are usually given the highest 2 thread priority levels to ensure it to run as quickly as possible.

With a SH3 reference platform running at 58.98 MHz internal and 14.745 external frequency, the typical latency of ISR is between 1.3 and 7.5 ms, and latency of IST is between 93 to 275 ms.

Linux implements threads in the kernel. Therefore, scheduling in LINUX uses threads as entities, not processes, in the kernel data structures. LINUX distinguishes three classes of threads for scheduling purposes, which are 1) Real-time FIFO threads are the highest priority and not preemptable. 2) Real-time round robin threads are the same as Real-time FIFO threads except for its preemptibility; 3) timesharing threads have lower priority than the previous two. Each thread has scheduling priority and a quantum associated with it. Linux schedules threads via a GOODNESS algorithm, which chooses to run the thread with highest goodness and the thread's quantum is decremented by one as it runs. Different from QNX and Window CE, LINUX, by nature, is not a fully preemptable design. When LINUX is ported to the embedded system, it suffers from several challenges to real-time applicability: determinism in general, and response under load in specific. Real-time operating system (RTOS) refers to an operating system that can perform in a predictable and repeatable manner, regard-less of workload. Therefore, a few efforts have been contributed to make LINUX a better RTOS. One idea is to create a RTOS kernel and running Linux as an uncheduled thread under the RTOS. In this dual structure Linux, high-priority threads that need true RTOS performance would run under the host operating environment at a higher priority than would the Linux thread. These real-time Linux kernel (RTLinux) provides a nominally generalized RTOS API and threading model, while relegating the main Linux kernel to the status of a low priority (idle) thread under their control. The "normal" Linux applications and RTLinux/RTAI threads communicate through shared memory and a limited set of other, mostly *ad hoc*, IPCs. RTLinux applications are written with the same structure as Linux kernel modules, that is, with potentially full access to Linux internals and with full benefit of the memory-protected process-based programming model. The second effort is to redefine the standard Linux scheduler and tune Linux device drivers, and hence to make Linux more responsive and deterministic. The first approach is to enhance the existing real-time policy in Linux threads repertoire by adding the real time priorities as referred as "real-time" scheduler. Therefore, this scheduler will select and dispatch the highest priority real-time process in deterministic time period. In order to make the Linux kernel to be fully preemptible, three modifications are to be made to the Kernel. First, the definition of the SMP kernel's primary IPC and the spinlock are changed from its SMP specific implementation to a preemption lock, which acts as a control on reentrancy to critical sections of kernel code. Second, the interrupt handling part is modified to allow rescheduling on return from interrupt if a higher priority process has become executable, even if the interrupted process was running in kernel mode. Moreover, SMP spin unlocks are also redefined to return the system to a preemptible state, and check if an immediate context switch is needed. Lastly, the kernel build definition for a uniprocessor target system is modified to include the spinlocks construct (implemented as preemption locks). An important issue in external interrupt management is the use of the "interrupt-off" processor mode which ensures the kernel not to be reentered for critical region. Another critical design issue for interrupt management in Linux is the ability of the system to "nest" interrupts—to accept a higher priority interrupt while still servicing an interrupt. If the interrupted process runs in user mode, the kernel can switch context immediately to the newly executable highest-priority process after exiting the interrupt service routine. A hybrid kernel is deployed to manage interrupt-off mode and the real time scheduling. While RTLinux is used as an underlying kernel technology, all non-time critical components can be standard Linux processes, Interrupt off/on operations by Linux are emulated by the lower-level kernel. In this fashion, Linux semantics are not impacted, while actual interrupts (particularly those handled by the lower-level kernel tasks) are not blocked.

4. Interprocess communication

QNX uses message-passing facilities for interprocess communication. Since these primitives copy data directly from process to process without queuing, message delivery performance approaches the memory bandwidth of the underlying hardware.

Because embedded system usually has no large amount of memory, efficient usage of memory is important for system performance. In other message-passing system, message usually occupies a contiguous area of memory and message data is copied between processes, while in embedded system these are not good ideas. QNX message passing primitives support multipart messaging, such that a message delivered from one process to another need not occupy a single, contiguous area in the memory. QNX uses an MX table to indicate where the sending and receiving message fragments reside in memory. This allows messages that have a header block separate from the data block to be sent without performance-consuming copying of the data to create a contiguous message.

All the system services within QNX are built upon these message-passing primitives. Some will consider that message passing mechanism has more overhead than the monolithic kernel. Two concepts in QNX contribute significantly to overall system performance. One is the support for interrupt handlers directly within resource managers; the other is the multipart message passing primitives.

Windows CE support message passing between processes, it also support memory mapping between processes. This results in very fast data transfers between cooperating processes. It can be used to dramatically enhance real-time performance.

Embedded Linux uses original Linux IPC mechanisms. The Linux IPC mechanism is provided so that concurrently executing processes have a means to share resources, synchronize and exchange data with one another. Linux implements all forms of IPC between processes executing on the same system through shared resources, kernel data structures, and wait queues. Linux provides the following forms of IPC:

- Signals – perhaps the oldest form of Unix IPC, signals are asynchronous messages sent to a process.
- Wait queues – provides a mechanism to put processes to sleep while they are waiting for an operation to complete.
- File locks – provides a mechanism to allow processes to declare either regions of a file, or the entire file itself, as read-only to all processes except the one which holds the file lock.
- Pipes and Named Pipes – allows connection-oriented, bi-directional data transfer between two processes either by explicitly setting up the pipe connection, or communicating through a named pipe residing in the file-system.
- System V IPC: Semaphores, Message queues, Shared memory – a mechanism by which several processes have access to the same region of physical memory.
- Unix Domain sockets – another connection-oriented data-transfer mechanism that provides the same communication model as the INET sockets, discussed in the next section.

The embedded Linux can choose any one of the IPC methods for its particular application.

5. Memory management

Most modern conventional OS uses paged Virtual memory. Virtual Memory page is the unit of protection and memory allocation. The use of Virtual memory is also closely related to multiprogramming, because strictly speaking, each process operates in its own environment and address space; each process has its own virtual memory mapping, thus its own page table, upon process switching, different page tables are used.

The use of processes and memory protection in embedded systems is very important to embedded system. If a single address space is used for all applications, a software bug in a single application can result in

corruption of the memory and leads to the failure of the system. The disadvantage, however, is that the memory protection requires the CPU to support a MMU, which result in a more complex CPU, and the overhead of context switching between processes can be high.

Different from conventional OS, most embedded OS'es are targeted to simple CPU which often don't have MMU, have limited memory, little or no disk, so they often don't use virtual memory or use restricted Virtual memory, Which we'll see below in the examples of WinCE and uClinux.

Microcontroller Linux (μ Clinux) is a port for architecture without MMU, running on a flat memory model. The core kernel is used intact with a "virtual = physical" MMU model implemented in place of the address translation model. The official documentation from www.uClinux.org doesn't specifically say that there's no paging in uClinux, but the FAQ section made it very clear that:

1) there's no paging, data, text, stack memory area are contiguous, there's no way to automatically grow the stack. And a look into the code mm/memory.c shows that it actually copies a memory area on inability to allocate a large enough area.

2) conventional Linux uses sbrk() to implement malloc(), but there' no sbrk() in uClinux, you have to use mmap() to allocate heap(), which is implemented very simplistically.

3) there's absolutely no protection, the stack can grow into text or data , one process can read or write the data of another process.

4) since there's no paging, there' no memory sharing. While the fork() system call is implemented using copy-on-write(), there's no way to implement copy-on-write without sharing, so there's no fork() in uClinux, only vfork(), and the limitation of vfork() is that the parent blocks until the cild does exec() or exit(). The programmer has the responsibility of not changing the data , which is shared by both the parent and child.

On the other hand, Windows CE has more elaborate memory management, it supports paged virtual memory partially: It requires the CPU to support a TLB, however, a full page model - reading and writing pages of memory to a paging file on a backing store devices - is not supported. Windows CE support both processes and threads. Full memory protection applies to application processes.

6. Network support

Network support is important to embedded system because it makes them easily communicate with the outside world also easy to upgrade them. Almost all embedded system OS support network facilities in their kernel.

QNX contains low-level network communication in its micro kernel.

Windows CE has communication stacks of various kinds at the same level as kernel. It supports Internet Protocol (IP). It also provides a PPP layer underneath the IP stack, allowing the system to communicate over a serial cable or modem to a desktop machine. Additionally, Windows CE provide an IrDA 1.0 stack to allow infrared connections at speeds up to 115kbps. For communication APIs, Windows CE provide WinINET, with support for FTP, HTTP, and Remote Access Client, for connecting to Remote Access Server.

Linux also has very good network support. Embedded Linux directly benefits from the leading-edge support that Linux server and client-operating environments provide for networking stacks and Internet protocols. This means that embedded developers using Linux automatically get the most current Internet protocol support by default. The technologies for efficient, effective embedded networking are readily available. By carefully paring down the possibilities to a subset that is relevant to embedded systems, design, implementation, installation, and administration can all be greatly simplified.

Conclusion

As we have seen, the operating system design of embedded system is greatly impacted by both the application requirement and hardware limitation.

Application requirement impact

1. The embedded operating system scheduling, process management, protection system design need to take into account the embedded application attributes (e.g. real-time, relatively few processes coexisting, or even single task, fixed application)
2. The scalability and flexibility of embedded application is also an important issue that system designer must consider in choosing the system architecture.

Hardware Impact

Hardware platform features (e.g. lack of hard disks, small memory, no MMU support, low power consumption) force the designer to pay special attention to the efficient usage of system resource.

Upon the considerations listed above, current embedded system adopted the following design strategies: Currently embedded operating systems are targeted for CPU architectures ranging from x86 (CISC) to ARM, with very limited resources. They are usually designed in modular fashion or take micro-kernel structure to be flexible and optimally use hardware. Process management becomes relatively easy because the number of processes is relatively small, but scheduler is often required to satisfy real-time behavior, usually by setting high priority. IPC uses various mechanisms, the message passing passes only pointers to minimize memory usage and reduce copying. Memory management uses simplified paged virtual memory or even flat memory because MMU is simple, or even non-existent, and swapping is supported. Since mobile computing has become a trend, and most hardware have built-in network interface, network is supported in all of these systems.

There seems an ongoing trend to make the embedded device to be reusable and reconfigurable. From the hardware perspective, the CPU can be configured at real time. For example, the bit width of the operand can be dynamically changed to reduce the power consumption. There is demand for new embedded operating system to fit into the reconfigurable hardware platform. From the software perspective, the embedded operating system can be divided into two separated parts. One remains unchanged within the embedded system, while the other can be downloaded or upgraded on demand. Embedded Java is going towards this trend.

Reference

- [1] Gary Robertson, "Migrating Application from Solaris to Linux", MontaVista Software
- [2] Bill Weinberg, "Rediscovering Unix Reliability in Linux", MontaVista Software
- [3] Jun Sun, "Interrupt Latency Measurement Tool for Linux," MontaVista Software
- [4] Kevin D. Morgan, "Linux for Real-Time Systems: Strategies and Solution", MontaVista Software
- [5] Bill Weinberg and Claes Lundhlm, "Embedded Linux – Ready for Real-Rime", MontaVista Software
- [6] Jim Ready and Bill Weinberg, "Leveraging Linux for Embedded Applications", MontaVista Software
- [7] D. Hildebrand. "An architectural overview of QNX". In 1st USENIX Workshop on Microkernels and Other Kernel Architectures, pages 113—126.
- [8] Joel R. Williams, "Embedding Linux in a Commercial Product ", Linux Journal, Oct. 1999
- [9] ---, "BlueCat Linux Kernel Porting Guide" , Lynuxworks.com
- [10] Joel Williams , "The case for embedded Linux", Emlinux
- [11] JoedeBlaquiere,uclinux.orgattle, "Supporting New Hardware Environments with uClinux ", WA, April 1992.
- [12] Rex Page, " Windows CE in Embedded Applications," Embedded systems conference, 1998.
- [13] Sharad Mathur, "Real-Time Systems with Microsoft Windows CE", Embedded systems conference, ---.
- [14] Paul Yao, "What is Windows CE ?" Embedded systems conference, ----.
- [15] Jeff McLeman,"Incorporating Windows CE into an Embedded System from Start to Finish" Embedded systems conference, ----.