International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 18 (2017) pp. 7976-7981

An Analysis of Embedded Operating Systems: Windows CE, Linux, VxWorks, uC/OS-II, and OSEK/VDX

Sukhyun Seo, Junsu Kim, *Su Min Kim

Department of Electronics Engineering, Korea Polytechnic University, 15073 Siheung, Republic of Korea.

**Corresponding author*

Orcid: 0000-0003-1800-3130, 0000-0001-9583-7978, 0000-0002-5951-9208

Abstract

An embedded system is designed for a dedicated purpose which is different from the intended general usage intended for a desktop computer and a workstation. A simple and low cost embedded system can be designed without using embedded operating systems (OS). Considering the design time and effort, however, better design of embedded systems can be achieved using an embedded OS. With specific given constraints or a specific purpose for the embedded system, one embedded OS can save more time, cost, and effort in designing this embedded system than another embedded OS. This paper compares five embedded OS: Windows CE, Linux, VxWorks, uC/OS-II, and OSEK/VDX. It discusses several issues of embedded OSs such as process scheduling, memory management, and network support.

Keywords: embedded system, operating system, real-time, Windows CE, Linux, VxWorks, uC/OS-II, OSEK/VDX

INTRODUCTION

The constraints and purposes of embedded systems are different from each other. An appropriate software architecture must be selected for a given embedded system. There are two common types of software architectures to design embedded systems. One is to design these systems without using an embedded OS. In this design, the software is built as a super-loop structure. The loop calls subroutines and each subroutine manages some hardware or software. This software architecture is better to design a simple embedded system. The other is to design embedded systems using an embedded OS. In this design approach, an embedded OS can provide most of the basic functions. Since the characteristics of embedded OSs are quite different, one embedded OS may be better than another in designing a given embedded system. Therefore, we should understand the characteristics of each of some important embedded OSs in depth in order to make good selections.

Five embedded OS are analyzed and compared from section 2 to section 5 in the following points: architecture, process handling, memory management, and network support. Section 6 concludes this paper.

ARCHITECTURE

The OSs, VxWorks, uC/OS-II, and OSEK/VDX, are used for hard real-time systems [1, 2] while Windows CE and Linux are used for soft real-time systems. Therefore, VxWorks, uC/OS-II, and OSEK/VDX support only multi threads while Windows CE and Linux support both multi processes and multi threads.

In the multi thread model, the threads can access the common memory area freely. Since application programs share the same address space with the OS kernel, they can use kernel resources. Therefore, the context-switching time is very fast. Since OS kernel and application programs are unified, all components are compiled together when the OS image is built. The multi thread model is suitable for a small and hard real-time embedded system. However, a trivial bug can stop the whole system. When using these OSs, application programs cannot be added to an embedded system after porting is done.

When using a multi process OS, application programs can be added to an embedded system even after porting is done. Windows CE can use Active-Sync function and Linux can download program files through a network or a serial cable to add application programs. In the multi process model, an application program is running in a different address space from the kernel address space. Different from the multi thread model, a bug in an application program cannot stop the whole system. The multi process model is widely used for a complex embedded system that needs various application services [3].

As shown in Figure 1, the architecture of Windows CE [4] is hierarchical. The HAL(Hardware Abstraction Layer) and OAL(OEM Adaptation Layer) lie at the bottom. The original equipment manufacturers (OEM) include them as part of Windows CE porting. Above them lie the Graphics Windowing and Events Subsystem (GWES), the kernel itself, and the communication stacks. The Remote API (RAPI) 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. Each application program runs in its own address space and interacts with the rest of Windows CE via the Win32 system call interface.

Linux [5, 6, 7] has a layering structure. It consists of HAL, the

Linux kernel, and several modules as in Figure 2. The HAL has hardware dependent code to support a wide range of processors. The Linux kernel consists of five subsystems: the process scheduler, the memory manager, the virtual file system, the network interface, and inter-process communication. Since the Linux kernel is monolithic, it consists of one layer in itself. If any subsystem is modified, whole kernel must be built again. But, device driver modules can be attached dynamically.

As in Figure 3, VxWorks [8, 9, 10] comes with kernel, board support pakages, and a middleware for device drivers. The operating system supports useful application support such as file system, network stack, and inter-process communications. The OS kernel is separate from middleware, applications and other packages, which enable easier bug fixed and testing of new features.

The OS, uC/OS-II [11], is a very small micro-kernel real-time OS. It has only basic OS services such as task management

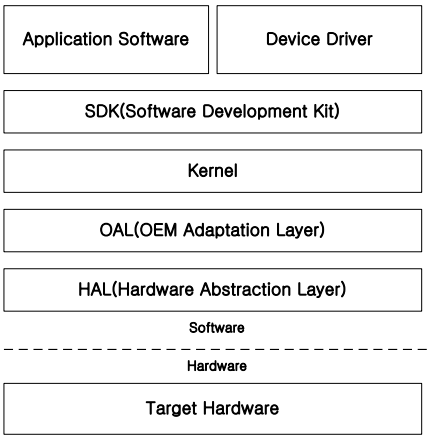

Figure 1. Windows CE architecture.

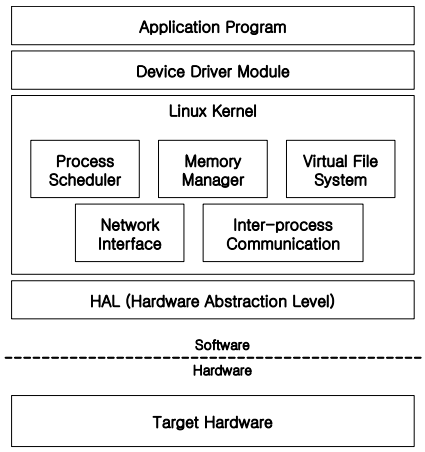

Figure 2. Linux architecture.

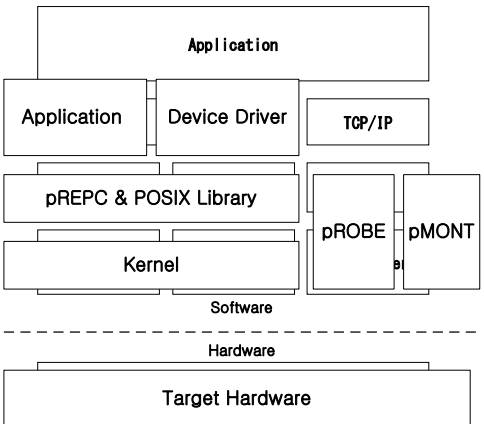

Figure 3. VxWorks architecture.

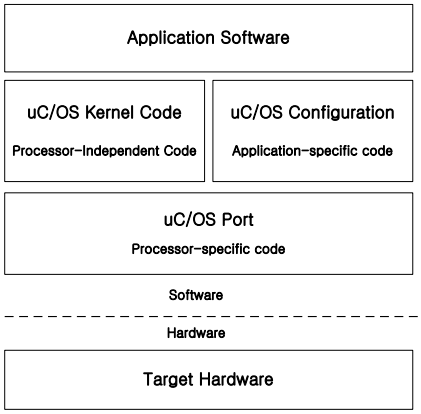

Figure 4. uC/OS-II Kernel architecture.

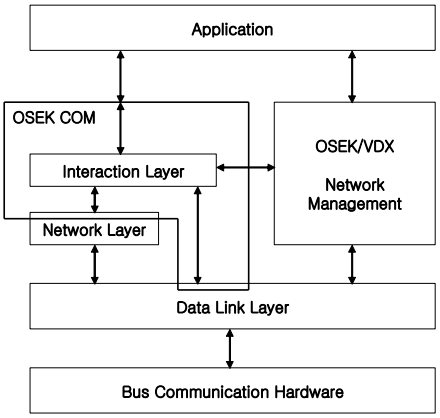

Figure 5. Layer model of OSEK/VDX with OSEK OS

services, time management services, inter task communication (ITC) services, and memory management services. Figure 4 shows the structure of the uC/OS-II kernel.

Figure 5 shows a layer model of OSEK/VDX with the OSEK OS [12, 13, 14]. The interaction layer provides the API, OSEK/VDX COM, to support the transfer of messages within and between network stations. For network communication, the interaction layer uses services provided by the lower layers. The network layer provides a protocol for the unacknowledged and segmented (USDT) transfer of application messages. The data link layer defines an open interface so that the OSEK/VDX protocols can interface with different type of buses. ECU-internal communication is handled by the interaction layer only.

PROCESS

Windows CE is a preemptive multitasking OS that supports processes and threads. Preemption conforms to only thread priority. A higher priority thread preempts a lower priority thread when it is ready to run. The same priority threads are scheduled according to a round-robin method. The default thread quantum is 100ms configurable by an OEM in the OEM adaptation layer (OAL). Multiple threads support 256 thread priorities. There are no more than 32 processes in one Windows CE system. However there is no theoretically limit on the number of threads in the system. Practically, the total number of threads is determined by available RAM resources.

The older version of the Linux kernel was not preemptive and did not support real-time. But Linux kernel 2.6 is a completely preemptive multitasking OS and supports real-time property. Linux also supports processes and threads. The Linux kernel 2.6 introduces new O(1) scheduler. The scheduler of this Linux kernel can perform scheduling in constant time. The Linux kernel distinguishes between one more type of process, a real-time process. The real-time process exists outside of the scheduler logic and always has highest priority. If processes have the same priority, they are scheduled by round-robin scheduling. The Linux kernel support dynamic priority-based scheduling for the real-time property.

The OS, VxWorks, is a preemptive multi-tasking OS. VxWorks has two major scheduling algorithms such as priority scheduling and round round-robin scheduling. In priority scheduling gives a priority to each process (thread). The highest priority thread is to be executed first and so on. The same priority processes are executed on a first come first serve basis. Priority can be decided based on time requirement, memory requirement or any other resource requirement. The Round-Robin is the scheduling algorithm used by the CPU during execution of the process. It is specially designed for time sharing systems. In round-robin scheduling assigned a fixed time to each process. Mean once a process is executed in a given time period then other process is permitted to execute for a given time period. The programmer defines the Task ID. In the user's point of view, the task's name is a unique ID identifying a task.

The OS, uC/OS-II, is also a preemptive multi-tasking OS. Each task has its unique priority and the highest priority task always runs first. uC/OS-II does not support round-robin scheduling. The OS, uC/OS-II, supports only the priority-based scheduling method. The maximum number of tasks is 64. Since tasks from priority 0 to 3 and from priority 60 to 63 are reserved for the OS, there remain 56 available tasks. Different from VxWorks, a user defines a uC/OS-II's task ID and a task does not have its name. Since the priority of each task is the same as its ID, the same task IDs cannot be used.

Two different task concepts are provided by the OSEK/VDX OS. A task is either basic or extended. It is either preemptive or non-preemptive. It has a statically defined priority and can suspend execution while waiting for an event. The combination of these and other attributes creates a conformance class. The task switching within OSEK/VDX OS is performed by a scheduler using one of three possible policies: non-preemptive, fully preemptive, or mixed preemptive. The non-preemptive scheduling policy of the OSEK/VDX OS is invoked in an application in which none of the tasks are configured as preemptive. Next, a fully preemptive scheduling policy is invoked in an application where all of the tasks are configured as preemptive tasks. Finally, the mixed preemptive scheduling policy is invoked in an application where some tasks have preemption enabled and other tasks have preemption disabled. Typically, an application operates in a mixed preemptive mode if most tasks can be preempted safely; however, a few tasks must run to completion without being preempted.

Every OS has a proper scheduling method for a real-time system. If hard real-time properties are needed, VxWorks, uC/OS-II and OSEK/VDX are suitable. If the system has to support multimedia or a heavy load, Windows CE and Linux are more suitable.

MEMORY MANAGEMENT

Windows CE uses paged virtual memory that is the unit of protection and memory allocation. The use of virtual memory is closely related to multi-programming. Each process operates in its own environment and address space. Since each process has its own page table, a different page table is used on process switching. Due to memory protection among processes, Widows CE requires its processors to have MMU (memory management unit). This results in more complex processors [2, 3]. Windows CE provides only one virtual addresses space of 4 GB for all the applications to use. The system still maintains protection between processes. The virtual memory space is split between Kernel-mode and User-mode. Kernel space is all addresses at or above 2G. The user space is below 2G. The Kernel space uses only Kernel-mode code with privileged

access and mostly static mapped virtual addresses. User space is organized as 64 slots of 32 MB; each process gets one slot. The maximum memory of an application is 32 MB.

The Linux kernel manages memory through the Memory Manager Subsystem illustrated in Figure 2. The Linux kernel also supports virtual memory management with a MMU and also physical memory management with a MMU-less processor. The memory management strategy depends on properties of the MMU. For example, paging memory allocation with the Buddy algorithm is used in ARM-compatible processors. Kernel space is above 1GB; and, user space is below 3GB. Dynamic memory allocation and dynamic stack size are supported. For a MMU-less processor, some parts of the Linux kernel are re-implemented. The fork, brk, malloc and some kernel functions are modified for the MMU-less processor; and, a flat-memory model is used. Then core part of physical memory management is in the HAL layer; and, the virtual memory manager, like the Slab Allocator, is in the kernel.

The OS, VxWorks, uses two conceptual ways to allot necessary memory to tasks. One is allotting various segments or memory that are not necessarily contiguous to a task. It allots the same memory amount that each task asks. While this is efficient memory utilization, it results in memory fragmentation problems and results in low system performance because of the complex inner mechanism. The other is allotting a multiple number of fixed amounts of memory. While this is not efficient memory utilization, it does not result in memory fragmentation problems. It does result in high sys-tem performance. As was explained previously, VxWorks does not support memory protection; and, every task share same memory address.

The OS, uC/OS-II, provides an integral number of blocks. One type of block exists or several types of blocks may exist. All blocks of each type have the same amount of memory. If several types of blocks are used, it does not make memory fragmentation problems. Each task in a uC/OS-II OS environment requires its own stack where the size may be different from another stack size. This can reduce the amount of memory needed in applications.

The OSEK/VDX OS specification does not support memory management functionality at runtime. The complete memory is allocated statically at system configuration time. The OS objects do not share common memory areas. This restriction does not allow the most efficient usage of RAM. It does greatly increases system stability and reduces the runtime overhead of the OS.

NETWORK SUPOORT

Windows CE supports the SNMP (Simple Network Management Protocol), TCP/IP, NDIS (Network Driver Interface Specification), and WinInet that is used to access HTTP service and FTP service in an Internet application program. Windows CE includes support for native Wi-Fi wireless local area networks (WLANs), Bluetooth, IrDA and ICS(Internet Connection Sharing). Windows CE 5.0 adds support for Dynamic Host Configuration Protocol for IPv6, called DHCPv6 Lite, which passes configuration data to devices in a TCP/IPv6 network.

Since Linux supports most protocols used in the Internet, it can support various functions needed in a web server, an E-mail server, a DHCP server, a DNS server, and a FTP server. Linux supports a network file system for Unix such as NFS or AFS. It can also exchange data with a Windows system because it supports Samba. Also, Linux supports most wireless network protocols, just as Windows CE, such as WLAN and IrDA.

The OS, VxWorks, has a stand-alone network protocol stack. xWorks includes drivers that support network connections over serial lines (using SLIP or CSLIP) or Ethernet networks (IEEE 802.3). It also supports connections over a backplane bus using shared memory. The VxWorks network stack uses the Internet protocols, based on the 4.4 BSD TCP/IP release, for all network

Table 1: The characteristics of each OS

	Windows CE	Linux	VxWorks	uC/OS-II	OSEK/VDX
IDE support	Yes	No	Yes	No	Yes
Support CPUs	32Bit (MMU)	16Bit or Higher	16Bit or Higher	8Bit or Higher	8Bit or Higher
Memory	350KB	125 - 512KB	60- 100KB	3-5KB	8-512KB
GUI support	Yes	Yes	No	No	No
Network support	Very Good	Very Good	Good	No	Very Good
User friendly	Very Good	Good	Good	POOR	Good
Real Time	Soft	Soft	Hard	Hard	Hard
System modification	No	Yes	No	Yes	Yes

communications. In addition to the remote access provided by Tornado, VxWorks supports remote command execution, remote login, and remote source-level debugging. VxWorks also supports standard BSD socket calls, remote procedure calls, SNMP, remote file access, boot parameter access from a host, and proxy ARP networks.

Since uC/OS-II does not have a network function, its designer must implement the desired network function by software or hardware.

Since the Network Management (NM) Components of OSEK/VDX was originally developed for an automotive environment, the network is assumed to be static. This means that the total set of possible nodes on the network is fixed and known by every node on the network. The OSEK/VDX OS's NM supports protocols dedicated for an automotive network such as CAN and LIN.

CONCLUSION

Usually, an embedded system has limited resources. Therefore, embedded OSs must be able to operate within the environments of these limitations. This paper analyzed and compared five OSs that can be used in embedded systems. As shown in Table 1, since the characteristics of these OSs are different from each other, designers must select an appropriate OS for the desired embedded system.

Since Widows CE provides very good IDE tools, various OS services, and a GUI (Graphic User Interface), it is suitable for designing a high-performance multimedia product. However, Windows CE requires high-performance 32-bit processors that must include MMUs. Windows CE is not suitable for a time critical application because it cannot support hard real-time.

Linux supports more processors than Windows CE. Since the Linux kernel source code is open, it is quite flexible. The release of Linux kernel 2.6 is more stable; and, this release supports soft real-time. However, the development environment of Linux is not as good as Windows CE.

Since VxWorks is a hard RTOS, VxWorks is suitable for a time critical application. Compared with Windows CE or Linux, VxWorks requires much simpler hardware. It is easy to develop device drivers. The development environment of VxWorks is good. However, VxWorks is not suitable for a graphical application because VxWorks does not support a GUI. The VxWorks OS is extremely stable and provides time critical support. Thus it is used for satellite systems and automatic spacecraft.

Since uC/OS-II provides only kernel code, there are few OS services. Therefore, designers must develop many system programs. However, uC/OS-II requires minimum hardware. We can build an embedded system on a one-chip micro controller by using uC/OS-II. Also, since uC/OS-II is open, it has similar advantages to Linux. The uC/OS-II OS was developed for educating students. Its source codes are very easy and well-documented.

The OSEK/VDX OS is also a hard RTOS. It is designed to provide optimal timing performance and is capable of meeting the tough demands of any processor. OSEK/VDX OS is used in many car designs. Some vehicles have many ECUs for engine, body, steering, and other parts. Since ECUs are controlled in a vehicle-network and a vehicle is a critical system, the OS for an ECU must be time critical and very stable.

ACKNOWLEDGEMENT

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No.2017R1C1B1004322)

REFERENCES

[1] J. W. Liu, Real-Time Systems. Prentice Hall, 2000.

[2] G. C. Buttazzo. "Hard Real-Time Computing Systems:Predictable Scheduling Algorithms and Applications", Springer Science & Business Media, 2011.

[3] Bic, L.F. and A.C. Shaw, "Operating Systems Principles", PrenticeHall,2003

[4] C. M. Netter and L. F. Baceller, "Assessing the real-time properties of Windows CE 3.0," ISORC-2001 Proceedings of Fourth IEEE International Symposium 2001, pp. 179-184, 2001.

[5] G. Newell, "Analysis of the Top 10 Linux Operating Systems," [Online]. Available: http://www.everydaylinuxuser.com/2014/02/analysis-of-top-10- linux-operating.html

[6] T. Nakajima, M. Iwasaki and S. Ochiai, "Issues for making Linux predictable," Proceedings of the Applications and the Internet (SAINT) Workshops 2002, pp. 8-14, 2002.

[7] Ivan Bowman, Conceptual Architecture of the Linux Kernel," http://plg.uwaterloo.ca/~itbowman/CS746G/a1.

[8] http://en.wikipedia.org/wiki/VxWorks.

[9] Wind River VxWorks Platforms 6.9 documentation. [Online]. Available: http://www.windriver.com

[10] http://www.scribd.com/doc/3792519/ Vxworks-Vs-RTLinux.

[11] Jean J. Labrosse, MicroC/OS-II The Real-Time Kernel, CMP Books, 2002.

[12] OSEK/VDX Binding Specification 1.4.1,

"http://www.osek-vdx.org/mirror/Binding141.pdf," OSEK Steering Committee, 2003.

[13] OSEK/VDX Operating System Specification 2.2.3, "http://www.osek-vdx.org/mirror/ttos10.pdf," OSEK Steering Committee, 2005.

[14] X. Chen, W. Yuan, K. Zhao, "Research and application based on OSEK/VDX OS," Proceeding of International Conference Computer, Mechatronics, Control and Electronic Engineering (CMCE), 2010.