International Journal of Engineering Research and General Science Volume 4, Issue 2, March-April, 2016
ISSN 2091-2730

# Hardware architecture of a Real Time Operating System

Shweta Ohri

Jagannath International Management School,

Vasant Kunj, New Delhi-70

Emails: shweta.ohri@jagannath.org

**Abstract-** In this paper we discuss prescribed design of a real time operating system. A real-time operating system provides a platform for the design and implementation of a wide range of applications in real-time systems, embedded systems, and mission-critical systems. The conceptual model of the real time operating system is introduced as the initial requirements for the system. Our main focus on this paper is to explain some fundamental processes of real time operating system like The CPU Scheduling Process, The System Initialization Process, The System Clock Process, and The System Event Capture Process in brief.

**Keywords:**

*The Real Time Operating System, The CPU Scheduling Process, The Memory Management Process, The Device Management Process, The File Management Process, UPMS for process management in real time operating system, UPMs for System Management in real time operating system.*

## 1. Introduction

As we know that An operating system is a set of integrated system software that organizes, manages, and controls the resources and computing power of a computer or a computer network. It also provides users a logical interface for accessing the physical machine in order to run applications. A general-purpose operating system may be perceived as an agent between the hardware and resources of a computer and the applications and users. An operating system may be divided into three subsystems known as those of the kernel or system management, the resource management, and the process management The kernel of an operating system is a set of central components for computing, including CPU scheduling and process management. The resource management subsystem is a set of supporting functions for various system resources and user interfaces. The process management subsystem is a set of transition manipulation mechanisms for processes and threads interacting with the system kernel and resources.

A real-time operating system (RTOS) is an operating system that supports and guarantees timely responses to external and internal events of real-time systems. An RTOS monitors, responds to, and controls an external environment, which is connected to the computer system through sensors, actuators, or other input-output (I/O) devices. In a real-time system in general and an RTOS in particular, the correctness of system behaviors depends not only on the logical results of computation but also on the time point at which the results are obtained. Real-time systems can be divided into hard and soft real-time systems. In the former, a failure to meet timing constraints will be of serious consequences, while in the latter, a timing failure may not significantly affect the functioning of the system.

A great variety of real-time operating system has been developed in the last decades. The existing real-time operating system is characterized as target-machine-specific, implementation-dependent, and not formally modeled. Therefore, they are usually not portable as a generic real-time operating system to be seamlessly incorporated into real-time or embedded system implementations. Problems often faced by real-time operating system are CPU and tasks scheduling, time/event management, and resource management. Efficient and precise methodologies and notations for describing solutions to these problems are critical in RTOS design and implementation. Generally, real-time operating system requires multitasking, process threads, and a sufficient number of interrupt levels to deal with the random interrupt mechanisms in real-time systems. In modern real-time operating system, multitasking is a technique used for enabling multiple tasks to share a single processor. A thread is individual execution of a process in order to handle concurrent tasks. In addition, an interrupt is a request of an external device or internal process that causes the operating system to suspend the execution of a current low priority task, serve the interrupt, and hand control back to the interrupted process.

This paper develops a comprehensive design paradigm of the formal real-time operating system in a top-down approach. In the remaining of this paper, there are five more sections. In Section 2, we provide a succinct thought about the CPU scheduling process.

Also, the memory management process, the device management process and the file management process are discussed in section 3, 4 and 5. Finally, conclusions and references are provided in Section 6 and 7.

## 2. The CPU Scheduling Process

CPU scheduling is the most inner kernel process of a real-time operating system to maximize CPU utilization. Real time operating system adopts multiprocess and multithread techniques to keep the CPU running different processes or threads of multiple processes on a time-sharing basis with a predetermined scheduling interval. The CPU scheduler selects which process in the ready queue should be run next based on a predefined scheduling algorithm or strategy. The CPU scheduler switches control of the CPU to the process selected for a certain time interval.

In real-time operating system, every ten time-intervals (10ms) forms a cycle to run all ready processes in the ReadyQHST (maximum 6) and ReadyQLST (maximum 4) as specified in the design of the system control block SCBST. The CPU scheduler process of real-time operating system, CPUSchedulerPC, dispatches a maximum of ten processes in both ready queues ReadyQHST and ReadyQLST within a low interrupt period 10ms, where each of them will be dispatched for running for 1ms scheduled by the high interrupt period. In the peak CPU load period of real-time operating system, 6 tasks from ReadyQHST and 4 from ReadyQLST will be scheduled, respectively, in a series of ten 1ms time intervals. However, during low CPU load period, any combination of high/low tasks may be scheduled including a series of multiple threads of the same process. In case there is a current task in the CPU but no other task in the ReadyQHST and ReadyQLST, the currently scheduled task may continue to run for at least another 1ms interval until the statuses of the ready queues are changed. Except the above two special conditions, the current process will be swapped out of the CPU as a delayed process. Instead, a new qualified process in the front of ReadyQHST or ReadyQLST will be scheduled into CPU for running.

## 3. The Memory Management Process

Memory management is one of the key functions of operating systems because memory is both the working space and storage of data and files. real-time operating system adopts a flexible segmentation method for system memory management with variable sizes of memory blocks for different events and processes. MemoryManagementPC supports dynamic and absolute physical memory manipulations that are necessary for real-time operating systems and real-time applications.

The process AllocateMemoryPC deals with dynamic memory allocations in real-time operating system. When a request for memory from a process is identified by the system, AllocateMemoryPC searches for a suitable sized memory block that is free. When a suitable memory block is identified by a given memory block number (MBN), AllocateMemoryPC sets its status as in-use.

The process ReleaseMemoryPC deals with dynamic memory release requests of the system when a process terminates and the memory allocated to it is no longer required. ReleaseMemoryPC operates on the memory control block when it is invoked. It gets the memory block number associated to the process that requests to release the memory. Then, it sets the status of the given memory block to free.

## 4. The Device Management Process

Devices as well as their I/O ports are directly manipulated by the processes DeviceManagementPC in real-time operating system. DeviceManagementPC encompasses both GetDevicePC and ReleaseDevicePC processes. Additional plug-in device drivers may also be incorporated via the same interface mechanisms by the adding device and deleting device processes.

The process GetDevicePC handles device allocations in real-time operating system. When a request for a device from a process is identified by the system, GetDevicePC searches for a suitable type of device that is free. When a suitable device is found, GetDevicePC sets its status as seized. Then, the associate process required the device is registered in the device control block. If there is no free and/or suitable device available in the system, a feedback status, &DeviceAllocatedBL:= F, will be generated.

The process ReleaseDevicePC handles device release requests for the system when a process terminates the use of a device. ReleaseDevicePC operates on the device control block when it is invoked. It first gets the device number (DNN) to be released associated to the process. Then, it sets the status of the given device in DCBST as free and disconnects the link between the device and the previously associated process.

## 5. The File Management Process

On the basis of the architectural model of the file system FilesST, the behaviors of the file management subsystem of real-time operating system, FileManagementPC, are modeled by a set of 14 behavioral processes in the categories of file system administrations and file manipulations. .

A process is a basic unit of system function that represents an execution of a program on a computer under the support of an operating system. A process can be a system process or a user process. The former executes system code, and the latter runs an application. Processes may be executed sequentially or concurrently depending on the type of operating systems.

A thread is an important concept of process management in operating systems. A thread is a basic unit of CPU utilization, or a flow of control within a process, supported by a PCBST, a program counter, a set of registers, and a stack. Conventional operating systems are single thread systems. Multithreaded systems enable a process to control a number of execution threads. The benefits of multithreaded operating system are responsiveness, resource sharing, implementation efficiency, and utilization of multiprocessor architectures of modern computers.

According to the high level specifications of the static behaviors of real-time operating system, the process management subsystem of real-time operating system encompasses a set of nine task handling processes corresponding to the states of behavioral processes for the entire task lifecycle of process manipulating.

***The Static Behavioral Models of the Real Time Operating System***

According to the RTPA methodology, a static behavior is an encapsulated function of a given system that can be determined before run-time. On the basis of the system architecture specifications and with the UDMs of system architectural components developed in the preceding section, the operational components of the given real-time operating system system and their behaviors can be specified as a set of UPMs as behavioral processes operating on the UDMs.

The basic functions of operating systems can be classified as system management, resources management, and processes management. The high-level static behaviors of real-time operating system StaticBehaviorsPC, encompasses three process subsystems such as SystemManagementPC, ResourcesManagementPC, and ProcessesManagementPC in parallel.

The following subsections describe how the real time operating system static behaviors in the three subsystems are modeled and refined using the denotation mathematical notations and methodologies of RTPA.

***UPMs for System Management in Real-Time Operating System***

According to the high level specifications of the static behaviors of real time operating system, the system management subsystem of real time operating system encompasses a set of five behavioral processes such as system initialization, system clock manipulation, system event capture, Device interrupt handling, and the CPU scheduler.

## 6. Conclusion

The design of real-time operating systems has been recognized as a comprehensive and complex system design paradigm in computing, software engineering, and information system design. This paper has demonstrated that the real time operating system (RTOS), including its architecture, static behaviors, and dynamic behaviors, can be essentially and sufficiently described by real time process algebra. On the basis of the formal specifications of real time process algebra (RTPA) by the coherent set of unified data models (UDMs) and unified process models (UPMs) in RTPA, the architectural and behavioral consistency and run-time integrity of an RTOS have been significantly enhanced. It has been identified that RTOS can be applied not only as a formal design paradigm of RTOS's, but also a support framework for a wide range of applications in design and implementation of real-time and embedded systems. The RTOS model may have also provided a test bench for the expressive power and modeling.

With a stepwise specification and refinement methodology for describing both system architectural and operational components, the formal model of the RTOS system has provided a foundation for implementation of a derived real-time operating system in multiple programming languages and on different operating platforms. It has also improved the controllability, reliability, maintainability, and quality of the design and implementation in real-time software engineering. The formal models of RTOS have been adopted as part of the supporting environment for the implementation of the RTPA-based software code generator (RTPA-CG). On the basis of the formal and rigorous models of the RTOS, code can be automatically generated by RTPA-CG or be manually transferred from the formal models.

A series of formal design models of real-world and real-time applications in RTPA have been developed using RTPA notations and methodologies in the formal design engineering approach, such as the telephone switching system, the lift dispatching system, the automated teller machine (ATM), the real-time operating system, the air traffic control system, the railway dispatching system, and the intelligent traffic lights control system. Further studies have demonstrated that RTPA is not only useful as a generic notation and

methodology for software engineering, but also good at modeling human cognitive processes in cognitive informatics and computational intelligence as reported.

**REFERENCES:**

[1] Anderson, T. E., Lazowska, E. D., & Levy, H. M. (1989). The Performance Implications of Thread Management Alternatives for Shared-Memory Multiprocessors. IEEE Transactions on Computers, 38(12), 1631–1644. doi:10.1109/12.40843
[2] Bollella, G. (2002). The Real-Time Specification for Java. Reading, MA: Addison Wesley.
[3] Brinch-Hansen, P. (1971). Short-Term Scheduling in Multiprogramming Systems. In Proceedings of the Third ACM Symposium on Operating Systems Principles (pp. 103-105).
[4] Deitel, H. M., & Kogan, M. S. (1992). The Design of OS/2. Reading, MA: Addison-Wesley.
[5] Dijkstra, E. W. (1968). The Structure of the Multiprogramming System. Communications of the ACM, 11(5), 341–346. doi:10.1145/363095.363143
[6] ETTX. (2009, February). In Proceedings of the First Eueopean TinyOS Technology Exchange, Cork, Ireland.
[7] Ford, B., Back, G., Benson, G., Lepreau, J., Lin, A., & Shivers, O. (1997). The Flux OSKit: a Substrate for OS and Language Research. In Proceedings of the 16th ACM Symposium on Operating Systems Principles, Saint Malo, France.
[8] ISO/IEC 9945-1. (1996). ISO/IEC Standard 9945-1: Information Technology-Portable Operating System Interface (POSIX) - Part 1: System Application Program Interface (API) [C Language]. Geneva, Switzerland: ISO/IEC.
[9] Kreuzinger, J., Brinkschult, U. S., & Ungerer, T. (2002). Real-Time Event Handling and Scheduling on a Multithread Java Microntroller. Dordrecht, The Netherlands: Elsvier Science Publications.
[10] Labrosse, J. J. (1999). MicroC/OS-II, The Real-Time Kernel (2nd ed.). Gilroy, CA: R&D Books.
[11] Lamie, E. L. (2008). Real-Time Embedded Multithreading using ThreadX and MIPS.
[12] Laplante, P. A. (1977). Real-Time Systems Design and Analysis (2nd ed.). Washington, DC: IEEE Press.
[13] Lewis, B., & Berg, D. (1998). Multithreaded Programming with Pthreads. Upper Saddle River, NJ: Sun Microsystems Press.
[14] Liu, C., & Layland, J. (1973). Scheduling Algorithms for Multiprogramming in Hard-Real-Time Environments. Journal of the Association for Computing Machinery, 20(1), 46–56.
[15] McDermid, J. (Ed.). (1991). Software Engineer's Reference Book. Oxford, UK: Butterworth Heinemann Ltd.
[16] Ngolah, C. F., & Wang, Y. (2009). Tool Support for Software Development based on Formal Specifications in RTPA. International Journal of Software Engineering and Its Applications, 3(3), 71–88.
[17] Ngolah, C. F., Wang, Y., & Tan, X. (2004). The Real-Time Task Scheduling Algorithm of RTOS+. IEEE Canadian Journal of Electrical and Computer Engineering, 29(4), 237–243.
[18] Peterson, J. L., & Silberschantz, A. (1985). Operating System Concepts. Reading, MA: Addison-Wesley.
[19] Rivas, M. A., & Harbour, M. G. (2001). MaRTE OS: An Ada Kernel for Real-Time Embedded Applications. In Proceedings of Ada-Europe 2001, Leuven, Belgium.
[20] Sha, L., Rajkumar, R., & Lehoczky, J. P. (1990). Priority Inheritance Protocols: An Approach to Real-Time Synchronization. IEEE Transactions on Computers, 1175. doi:10.1109/12.57058
[21] Silberschatz, A., Galvin, P., & Gagne, G. (2003). Applied Operating System Concepts (1st ed.). New York: John Wiley & Sons, Inc.
[22] Tanenbaum, A. S. (1994). Distributed Operating Systems. Upper Saddle River, NJ: Prentice Hall Inc.
[23] Viscarola, P. G., & Mason, W. A. (2001). Windows NT Device Driver Development. New York: Macmillan.
[24] Wang, Y. (2002). The Real-Time Process Algebra (RTPA). Annals of Software Engineering: An International Journal, 14, 235–274. doi:10.1023/A:1020561826073
[25] Wang, Y. (2004). Operating Systems. In Dorf, R. (Ed.), The Engineering Handbook (2nd ed.). Boca Raton, FL: CRC Press.
[26] Wang, Y. (2007). Software Engineering Foundations: A Software Science Perspective. In Software Engineering, Vol. II. New York: Auerbach Publications.
[27] Wang, Y. (2008a). RTPA: A Denotational Mathematics for Manipulating Intelligent and Computational Behaviors. International Journal of Cognitive Informatics and Natural Intelligence, 2(2), 44–62.
[28] Wang, Y. (2008b). Mathematical Laws of Software. Transactions of Computational Science, 2, 46–83. doi:10.1007/978-3-540-87563-5_4
[29] Wang, Y. (2008c). Deductive Semantics of RTPA. International Journal of Cognitive Informatics and Natural Intelligence, 2(2), 95–121.
[30] Wang, Y. (2008d). On Contemporary Denotational Mathematics for Computational Intelligence. Transactions of Computational Science, 2, 6–29. doi:10.1007/978-3-540-87563-5_2
[31] Wang, Y. (2009a). Paradigms of Denotational Mathematics for Cognitive Informatics and Cognitive Computing. Fundamenta Informaticae, 90(3), 282–303.
[32] Wang, Y. (2009b). The Formal Design Model of a Telephone Switching System (TSS). International Journal of Software Science and Computational Intelligence, 1(3), 92–116.

[33] Wang, Y., & Chiew, V. (2010). On the Cognitive Process of Human Problem Solving. Cognitive Systems Research: An International Journal, 11(1), 81–92. doi:10.1016/j.cogsys.2008.08.003

[34] Wang, Y., & Huang, J. (2008). Formal Modeling and Specification of Design Patterns Using RTPA. International Journal of Cognitive Informatics and Natural Intelligence, 2(1), 100–111.

[35] Wang, Y., Ngolah, C. F., Ahmadi, H., Sheu, P. C. Y., & Ying, S. (2009). The Formal Design Model of a Lift Dispatching System (LDS). International Journal of Software Science and Computational Intelligence, 1(4), 98–122.

[36] Wang, Y., Ngolah, C. F., Zeng, G., Sheu, P. C. Y., Choy, C. P., & Tian, Y. (2010c). The Formal Design Model of a Real-Time Operating System (RTOS+): Conceptual and Architectural Frameworks. International Journal of Software Science and Computational Intelligence, 2(2), 107–124.

[37] Wang, Y., & Ruhe, G. (2007). The Cognitive Process of Decision Making. International Journal of Cognitive Informatics and Natural Intelligence, 1(2), 73–85.

[38] Wang, Y., Tan, X., & Ngolah, C. F. (2010b). Design and Implementation of an Autonomic Code Generator based on RTPA. International Journal of Software Science and Computational Intelligence, 2(2), 44–67