Design of High Trust Embedded Operating System Based on Artificial Intelligence

Xingjian Liu[a,*], Xiao Chen

Guangdong Business and Technology University, Zhaoqing, Guangdong, 526060, China

[a]starsword2@163.com

*Corresponding Author

Keywords: Operating System, Software Architecture, Lightweight, Multitask, Partition, Multilevel Security, Unified Architecture

Abstract: the Embedded Application Scenarios Are Very Extensive, Which Are Divided into Many Application Modes, Which Need Different Characteristics and Architectures of the Embedded Operating System. This Paper Analyzes the Mainstream Embedded Real-Time Operating System, Abstracts the Operating System into Light-Weight Operating System, Real-Time Multi Task Operating System, Partition Operating System, Multi-Level Security Operating System and Other Different Types According to the Application Mode and Technical Characteristics, and Describes the Software Architecture of the Operating System from the Perspective of Technical Characteristics, Capability Composition, Structural Framework, Etc., and Proposes a Unified and Open a Flexible Operating System Software Architecture Supports a Single Operating System to Meet the Requirements of Multiple Application Modes through Configuration.

1. Introduction

Embedded Systems Are Ubiquitous in Aerospace, Automotive Electronics, Network Products, Intelligent Home Appliances and Other Fields[1]. in These Systems, Embedded Operating Systems Are Usually Required to Manage Embedded Hardware and Software Resources. among Them, Some Systems Require Strict Real-Time Performance, Such as Control Equipment, Communication Equipment, Etc. in the Aerospace Field, and the Operating System Used is Required to Have Good Real-Time Performance, Which is Collectively Referred to as Embedded Real-Time Operating System (Rtos).

According to the specific application scenarios, embedded real-time operating system can be further divided into four categories: lightweight: for micro nodes, such as sensors, actuators, etc., with small-scale, low power consumption and other characteristics; real-time multitasking: for general scenarios, such as control, communication, display, etc[2].With good real-time, customizable; robust Zoning: for multiple application isolation requirements of integrated scenarios It has the characteristics of time / space isolation, health monitoring, etc. multi-level security: for scenarios requiring information protection capability, it has the characteristics of trusted kernel, security communication, etc.

There are some differences in the application scenarios, functional requirements and quality attributes of the above-mentioned embedded real-time operating systems. Currently, each type of operating system meets the specific use conditions and usually uses different software architectures. With the increase of operating system types and scale, a series of problems have emerged in software reuse, maintenance, certification and other aspects. Academia and industry began to explore an operating system software architecture, which can meet the needs of a variety of application modes.

2. Typical Embedded Real-Time Operating System Software Architecture

2.1 Overview of Embedded Real-Time Operating System Software Architecture

In the development of software intensive large-scale system or software product line with similar demand structure, it is necessary to consider the components that make up the system, the relationship or interaction between components, and the topological structure formed by the interaction between components[3]. As can be seen from the definition, software architecture consists of three parts: software components, relationships and constraints among components, and interconnected components to form the overall architecture of the system. Generally, the construction process of application system is[4]: problem definition, software requirements, software architecture, software analysis and design, software implementation, in which software architecture is the bridge between software requirements and software design. Because the application scenarios and problem-solving fields of all kinds of embedded real-time operating systems are relatively fixed, the architecture of each kind of embedded operating system usually has certain universality.

Table 1 Comparison of Features of Several Operating Systems

System characteristics	SOS	Manti OS	TinyOS
Programming mechanism	event driven	Thread driven	event driven
low power consumption	Processor energy management	Processor energy management peripheral energy management	Processor energy management peripheral energy management
task scheduling	Priority	Priority	FIFO
memory management	static state	static state	static state
System execution model	Modular	thread	assembly

For the embedded real-time operating system, the functional requirements usually include task management, communication management, time management, interrupt / exception and other basic capabilities, which may include partition management, power management, security components, network components and other functions depending on the specific application[5]. Non functional requirements have great differences for all kinds of operating systems, such as the operating systems used in safety critical areas focus on certainty, reliability, hardware compatibility, environmental compatibility, fault tolerance, health monitoring, certifiability, maintainability, reusability, etc. The above functional requirements and non functional requirements need to be considered in the software architecture design stage.

2.2 Architecture of Lightweight Operating System for Micro Nodes

The typical application scenario of this kind of operating system is wireless sensor. The characteristics of this kind of nodes are: sensor nodes have limited power energy, communication capacity and computing storage capacity; network has the characteristics of large-scale, self-organization, dynamic, reliability, etc. application relevance is strong. There are many embedded operating systems for wireless sensor networks, such as TinyOS, mantis OS, SOS, Contiki, vxmicro, etc.

The common software architecture design patterns of this kind of operating system include: Based on event triggering model, multi-threaded cycle scheduling model, and the difference between the above three software architecture design patterns is whether to introduce multi-threaded or preemptive. The operating system with event triggering and multithreading cycle scheduling is smaller in scale and simple in mechanism, which is commonly used in micro nodes with single functional requirements. For the software architecture supporting multithreading, multi priority and event management, the event driving and task execution are separated, and the preemptive multithreading scheduling mode is adopted. Such operating system is usually cut and performed by the general kernel Therefore, it has better scalability, better support for message management, storage management, module management, power management and other commonly used modules, with a wider range of applications.

2.3 Architecture of Real-Time Multitask Operating System for General Domain

Real time multitask embedded operating system is widely used in aerospace, industrial control, rail transit and other fields[5]. It usually supports priority based multitask, multi class communication, rich components, rich devices, rich interfaces, with good real-time, customizable and other features, such as common VxWorks, UC / OS, integrity, ose, QNX, RTEMS, etc.

With the improvement of application requirements, the function of the operating system is enhanced, the number of modules is increased, and the scale is enlarged. When carrying out the architecture design, more attention should be paid to the characteristics of the operating system, such as openness, scalability and security. Generally, the design strategies such as hierarchical, modular, object-oriented, standardized interface, polymorphism, space protection and other security strategies are adopted. Figure 1 shows the software architecture of a common real-time multitask operating system.

The early real-time multitask operating system does not support polymorphism, and the operating system and application run in the system state. With the improvement of hardware storage management unit (MMU) capability and the demand of isolation protection, the operating system gradually supports multi privilege state and task isolation.

2.4 Robust Partitioned Operating System Architecture for Application Integration

Integrated modular avionics system (IMA) can effectively reduce the volume, weight and cost of avionics system by integrating the application of multiple electronic devices in a combined environment[6]. At the same time, resource sharing also brings mutual interference between multiple applications. This kind of interference can be solved by the partition mechanism provided by the operating system. Through the time and space isolation of the application, the failure of one partition will not affect other partitions, which effectively supports system verification, validation and authentication. The avionics Technical Committee (AEEC) has issued a series of ARINC653 (aviation application software programming interface) standards, which specifies the programming interface provided by the operating system to the application:

Part 0 is the standard overview;

Part 1 is a required service;

Part 2 is extension service;

Part 3a is the required service compliance test specification;

Part 3b is the extended service compliance test specification;

Part 4 is a subset service;

Part 5 is the core software recommendation capability.

At present, the airborne operating system applied to ima conforms to ARINC653 standard and the software architecture specified in the standard. The operating system that meets the requirements of ARINC653 needs to provide the two-level scheduling capabilities of partition level and process level[7]. This kind of operating system can be divided into two layers: core layer and partition level. The core layer mainly realizes the functions of partition management, partition scheduling, health monitoring and partition communication. The partition level implements the application execution interface (APEX) that meets the requirements of ARINC653, including process management and process Communication, time management and other functions. The virtualization technologies such as system call and virtual interrupt are usually used to realize the information interaction between the two layers.

2.5 Multi Level Security Operating System Architecture for Information Security

In aviation, military and other security control systems, there are multi level security / safety (MLS) applications. Multilevel security system refers to a system that processes information of different sensitive levels (such as different security levels). Multilevel security system must be authenticated to ensure that it can process and output data of multiple security levels at the same time. Multiple independent levels of security / safety (mils) not only refers to the verifiable high assurance security architecture that implements different security level applications on a single kernel, but also refers to the security system design method that provides a reusable formal framework for high assurance system specification and verification. Mils combines the system operation security solution meeting FAA do-178b / C criteria and the information security solution meeting NAS / niap common criteria criteria. The generation of mils architecture is to simplify the design, analysis and verification process of high assurance system. It is a high assurance security architecture based on the "separation" idea proposed by rushby, and builds hierarchical security services through separation. The software architecture of mils is given. The separation kernel layer brings the most important security functions into the kernel to form trusted code, provides reliable partition and controllable information flow for middleware and applications, and middleware provides services such as resource allocation, partition communication, data subscription and publication. The application uses the separation kernel and security middleware to manage, control and execute specific security policies[8]. The following policies are required: non bypassable security function, small and verifiable security function evaluation, always - in - voked security function, tamperproof security function and data.

3. Definition of Unified Software Architecture of Embedded Real-Time Operating System

Because of the variety of application scenarios and use requirements of the embedded real-time operating system, and the different requirements for the technical characteristics of the operating system, there are many kinds of embedded operating system, and the architecture is not uniform. This situation brings a series of problems to operating system development: too many kinds of product lines, difficulty in software reuse, difficulty in software maintenance, high cost of development and certification. Therefore, the industry hopes to learn from the mode of general operating system such as windows or Linux, find a general embedded real-time operating system software architecture, and realize different operating system capability configurations through tailoring, configuration or combination, so as to meet the use needs of different applications.

In addition, with the rapid development of embedded system, embedded users pay more attention to the characteristics of embedded operating system, such as open architecture, scale expansion, capability customization, unified interface and so on. They tend to adopt unified software framework and shelf products (COTS) as much as possible to quickly build their own software products. In view of the common needs of operating system developers and application developers, a kind of operating system architecture that can meet a variety of embedded application scenarios has been explored at home and abroad.

With the improvement of computer technology and software technology, it is possible to meet the above requirements of a unified architecture embedded real-time operating system. Through the combination and optimization of various design strategies, a unified, open and elastic operating system software architecture can be defined. In the aspect of operating system design, the hierarchical implementation of hardware abstraction, kernel, component and application division, modular implementation of the definition of each module, module connection and constraint, micro kernel implementation of kernel high abstraction and minimum definition, extensible implementation of module dynamic addition and withdrawal, object-oriented implementation of object abstraction, data and service encapsulation, etc[9]. the implementation of multi scheduling policy framework is not Same as the support of scheduling method. In terms of development environment, it supports component-based management strategy, management of different types and versions of modules, and can be configured to generate operating systems with different capability configurations.

In the above architecture, the microkernel and the multi-mode scheduling extension module can not be tailored, and other modules can be combined or tailored according to the use mode requirements. The multi-mode scheduling extension module supports the policy configuration and execution of schedule scheduling, priority scheduling, cycle scheduling, etc.; the kernel auxiliary function extension module realizes the common auxiliary functions such as information browsing, equipment management, debugging support, etc.; the real-time process realizes the security isolation of the application tasks in the general field; the ARINC653 support module realizes the robust partition in the integrated scene; the security function realization The multi-level security capability meets the requirements of information security protection; the virtualization module provides different degrees of virtualization support capability for the upper application; the functional component is the file system, graphic image and other components configured according to different application requirements; the operation environment provides support services to the application according to different capability configurations. According to the above software architecture, the following capability configurations are formed for different application scenarios:

For micro nodes: micro kernel + multi-mode scheduling extension module (priority or event triggering);

For real-time multitasking: microkernel + multimode scheduling extension module (priority) + real-time process;

For integration: microkernel + multi-mode scheduling extension module (schedule scheduling) + ARINC653 support + operation environment;

For information security: microkernel + multi-mode scheduling extension module (time schedule) + security function + operation environment;

For hybrid scenarios (combined requirements of partition, information security and customer OS): microkernel + multi-mode scheduling extension module + ARINC653 support + security function + virtualization + operating environment.

4. Conclusion

In the future, the embedded application scenarios will be more diverse, which continuously puts forward new requirements for the embedded operating system. New features and new architecture of the embedded operating system will appear, such as the current operating system for the Internet of things, artificial intelligence, cloud computing and other new technologies is in the research hotspot. At the same time, reducing cost, improving efficiency and ensuring quality are the eternal pursuit. The academia and industry will also carry out the research and practice of embedded operating system with unified architecture for a long time, and successively launch customized operating system. At present, some embedded operating systems have carried out the practice of unified software architecture, the unity of multi task and lightweight, the unity of multi-level security and ARINC653, and will be unified in more types of operating systems in the future.

References

[1] Yue, Z., Yoshigoe, K., Jiang, B., et al. (2017). A Distributed Graph-Parallel Computing System with Lightweight Communication Overhead, vol. 2, no. 3, pp. 204-218.

[2] Zebin, Wu., Linlin, Shi., Jun, Li. (2017). GPU Parallel Implementation of Spatially Adaptive Hyperspectral Image Classification. IEEE Journal of Selected Topics in Applied Earth Observations & Remote Sensing, vol. 11, no. 4, pp. 1-13.

[3] A, Aimar., A, Aguado, Corman., P, Andrade. (2017). Unified Monitoring Architecture for IT and Grid Services. Journal of Physics Conference, vol. 898, no. 9, pp. 092033.

[4] Wei Wei, Dejun Jiang, Jin Xiong,. HAP: Hybrid-Memory-Aware Partition in Shared Last-Level Cache[J]. Acm Transactions on Architecture & Code Optimization, 2017, 14(3):1-25.

[5] Saveetha, V., Sophia, S. (2017). Optimization of K-Means Clustering on Graphics Processing Unit Using Compute Unified Device Architecture.

[6] Kyrill, Kunakhovich. (2018). What Remains: Everyday Encounters with the Socialist Past in Germany by Jonathan Bach (review). German Studies Review, no. 41.

[7] V, S, Sukhopluyeva., D, Y, Kuznetsov. (2017). Software system architecture for corporate user support[J]. Journal of Physics Conference, vol. 803, no. 1, pp. 012160.

[8] Wei, Yancong., Yuan, Qiangqiang., Shen, Huanfeng. (2017). Boosting the accuracy of multi-spectral image pan-sharpening by learning a deep residual network. IEEE Geoscience & Remote Sensing Letters, vol. 14, no. 10, pp. 1795-1799.

[9] Emmanuel, U., Ogbodo, David, Dorrell., Adnan, M. (2017). Abu-Mahfouz. Cognitive Radio based Sensor Network in Smart Grid: Architectures, Applications and Communication Technologies. IEEE Access, vol. 5, no. 9, pp. 19084-19098.