Operating System Development for Smart IoT Electricity Meters: Interpretation and Discussion of Application Processes

Authors: Zhang Chunhui, Zhang Zhen, Zhang Zhen

Date: 2024-01-26T00:00:00+00:00

Abstract

This article primarily focuses on the development and application of the operating system for State Grid intelligent IoT electricity meters, examining the key issues and challenges therein. It introduces design deficiencies in the new generation of smart electricity meters, attributing them mainly to the fact that their operating systems are developed by companies specializing solely in system development, which lack understanding of the products and their reliability, rendering them unsuitable for specialized power products such as electricity meters, concentrators, and fusion terminals. The article reviews the history of early operating systems applied in electricity meters and notes that these early systems, due to their high degree of generality, could not establish independent intellectual property rights. It further explores in depth the origin, characteristics, common versions, and architecture of the Linux embedded operating system, particularly its kernel, Shell, file system, and application programs. Additionally, the article references State Grid specifications regarding requirements for operating systems and application software, and elaborates on relevant terminology and definitions, including management module, plug-and-pull event, POSIX standard interface, application programs, and basic application programs. The article emphasizes that the application of embedded operating systems on electricity meters represents an innovation in electricity meter design technology, and proposes issues requiring discussion for the continued development of current intelligent IoT meter operating systems. In summary, this article provides a comprehensive and in-depth exploration of the development and application of the State Grid intelligent IoT electricity meter operating system, identifying the key issues and challenges involved, which holds significant guiding importance for advancing the further development and application of smart electricity meters.

Full Text

Preamble

The Development and Application Process of the Smart IoT Meter Operating System: An Interpretation and Discussion

ZHANG Chunhui[1], ZHANG Zhen[2]
(1. State Grid Shandong Electric Power Co., Ltd., Jinan, Shandong 250001, China;
2. Huaneng Jinan Huangtai Power Generation Co., Ltd., Jinan, Shandong 250100, China)

Abstract: This article focuses on the development and application of operating systems for smart IoT electricity meters in State Grid, discussing the key issues and challenges involved. It introduces design defects in the new generation of smart meters, primarily arising from the fact that their operating systems are developed by specialized system companies lacking understanding of product requirements and reliability, making them unsuitable for specialized power products such as electricity meters, concentrators, and fusion terminals. The article reviews the history of early operating system applications in electricity meters and points out that early systems, due to their strong generality, could not establish independent intellectual property rights.

The paper delves into the origin, characteristics, common versions, and architecture of Linux embedded operating systems, particularly examining the kernel, Shell, file system, and applications. It also cites State Grid specifications regarding requirements for operating systems and application software, and explains in detail related terms and definitions such as management modules, plug-and-play events, POSIX standard interfaces, applications, and basic applications. The article emphasizes that applying embedded operating systems to electricity meters represents an innovation in meter design technology, and raises issues requiring discussion for the further development of current smart IoT meter operating systems. Overall, this article provides a comprehensive and in-depth exploration of the development and application of State Grid smart IoT meter operating systems, identifying key issues and challenges that are significant for guiding the further development and application of smart meters.

Keywords: Smart IoT meter, Operating system

In 2022, the authors of this paper, including Mr. Zhang Zhen, have been closely following new developments in the development and application of the State Grid smart IoT electricity meter operating system. This focus stems from three primary considerations. First, following the publication on June 2, 2022, of our article on Landis+Gyr's new E360 series smart meters (which employ IoT communication technologies), extensive discussions among experts revealed design flaws in domestic new-generation smart meters. The main issue is that the operating system adopted for State Grid smart IoT meters is developed by specialized system companies that lack understanding of the product and,

more critically, product reliability, making it unsuitable for specialized power products such as electricity meters, concentrators, and fusion terminals.

Second, although early electricity meters did employ operating systems, this practice was not continued. However, in early 2022, State Grid released the "Smart IoT Electricity Meter Function and Software Specification (Q/GDW 12180-2021)" (hereinafter referred to as the "State Grid 12180 Specification"), which for the first time introduced embedded operating systems for domestic smart IoT meters. This represents a new exploration in applying embedded operating systems to electricity meters in China.

Third, most current international and domestic embedded operating systems are general-purpose software products. A metering expert asked the authors: "If we want to promote a completely self-developed operating system, what suggestions do you have?" This highlights that applying specialized product-specific operating systems to electricity meters constitutes an innovation in meter design technology.

This paper focuses on the State Grid smart IoT meter operating system, summarizing and interpreting its development and application process, and raising issues requiring discussion for the further evolution of current smart IoT meter operating systems.

1. Starting with Operating System Applications in Power Terminals

1.1 Historical Context of Operating Systems in Power Terminal Products

Power terminal products have employed operating systems for many years, accumulating over a decade of development experience. Most meter enterprises have adopted international general-purpose embedded operating systems (such as Linux), refined them for power terminal development, and released several versions, yet have been unable to establish independent intellectual property rights.

1.2 Origin and Architecture of Linux Embedded Operating Systems

Origin and Evolution: In 1984, Richard Stallman initiated the GNU project (a recursive acronym for "GNU is Not UNIX"), a software system plan based entirely on free software, and formulated a General Public License. In 1991, Finnish university student Linus Torvalds developed the first software program on his Intel 386 computer, released the source code via the Internet, and named it Linux, thereby creating the Linux operating system.

Key Characteristics: Linux is characterized as free software with open source code, offering high performance, strong security, ease of customization and redevelopment, high interoperability, comprehensive multitasking capabilities, and

true 32-bit operation.

Common Versions: Linux exists in two primary forms: kernel versions and distribution versions.

System Architecture: The Linux operating system comprises four main components: the kernel, Shell, file system, and applications. The kernel, Shell, and file system form the basic operating system structure, enabling users to run programs, manage files, and utilize the system.

The **Linux kernel** serves as the core of the operating system with numerous fundamental functions. It manages system processes, memory, device drivers, file systems, and network systems, determining overall system performance and stability. The **Linux Shell** provides the user interface, offering an interactive interface for users to communicate with the kernel. It receives user commands and passes them to the kernel for execution, functioning as a command interpreter. Additionally, Shell programming languages possess many features of conventional programming languages, making Shell programs as effective as other applications. The **Linux file system** defines how files are organized on storage devices such as disks. Linux supports multiple popular file systems. **Linux applications** refer to the standard program sets included in Linux systems, comprising text editors, programming languages, XWindow, office suites, Internet tools, and databases.

In summary, Linux's structure consists of both source code and software programs. The software programs include the basic operating system (kernel, Shell, file system) and the application suite.

2. Requirements for Operating Systems and Application Software in State Grid 12180 Specification (Summary)

The smart IoT electricity meter consists of three components: a metering module, a management module, and an extension module. The operating system serves as a unified development platform for the functional software of the management module, while the metering and extension modules continue to use traditional software development approaches.

2.1 Terminology and Definitions Related to Operating Systems and Application Software

Management Module (State Grid 12180 Specification No. 3.2): Composed of a data processing unit, display unit, security unit, storage unit, and control unit, this module enables display, external communication, event recording, data freezing, load control, and other functions. It acts as middleware connecting the metering module and extension module.

Management Module Plug-and-Play Event (No. 3.57): An event recorded when the electricity meter's metering module detects the insertion

or removal of the management module. Note: The metering module should determine this event based on detected plug/unplug signals or changes in the management module's ESAM serial number.

POSIX Standard Interface (No. 3.60): Portable Operating System Interface, a collective term for a series of application software standard interfaces defined by IEEE for software running on various operating systems.

Application Software (No. 3.61): Software programs running on the operating system to implement specific functions.

Basic Application Software (No. 3.62): Application software that executes management module business functions such as energy freezing, demand measurement, event recording, and control.

System Management Software (No. 3.63): Application software that manages and monitors other electricity meter applications to ensure stable system operation, responsible for communication protocol parsing and message routing.

Security Service Software (No. 3.64): Application software that manages calls to the electricity meter's ESAM module, manages access authentication for external devices, and ensures secure system operation.

Extension Application Software (No. 3.65): Application software that executes functions beyond those required for the management module in this document, according to future business application needs.

Semaphore Shuffling Time (No. 3.70): Represents the time delay from when one task releases a semaphore to when another task waiting for the semaphore is activated.

White-Box Testing (No. 3.71): Coverage testing based on the internal logical structure of software to determine whether the actual operating state is consistent with the expected state, also known as structural testing, transparent-box testing, logic-driven testing, or code-based testing.

2.2 Management Module Functional Requirements

The management module is equipped with an operating system capable of running various application software, enabling it to store various frozen data, conduct external communications, and serve as a communication router for the metering and extension modules. Its specific functions (15 items) include: energy data, demand measurement, clock synchronization, tariff and time period functions, freezing functions, event recording, clearing functions, communication functions, signal output, display functions, monitoring functions, fee control functions, power preservation functions, security protection, and software requirements.

2.3 Operating System Architecture

The functional software for the electricity meter management module should be developed on a unified operating system platform. Based on the management module hardware platform, electricity meter software is divided into four layers according to differences in basic functions, operating environment, and operating permissions: boot layer, driver layer, system layer, and application layer.

The **application layer** implements the operation and management of electricity meter system application software, business application software, and extension application software. System application software implements message routing, module management, event notification, application management, upgrade management, security management, and access authentication. Business application software implements electricity meter business functions, while extension application software implements electricity meter extension functions.

The **system layer** implements embedded operating system multitasking scheduling, inter-task communication, memory resource allocation and protection. The system layer includes the kernel, components, and POSIX standard interfaces.

The **driver layer** implements main controller drivers and peripheral drivers, is compatible with multiple M4 core hardware platforms, and provides unified interfaces to the system layer and application layer.

The **boot layer** implements initialization of some hardware and guides kernel startup.

2.4 Operating System Requirements

Resource Requirements: CPU must adopt an architecture with performance no lower than Cortex-M4; memory must be no less than 512kB; FLASH capacity must be no less than 1MB.

Functional and Performance Requirements: - **Kernel:** Supports basic functions including memory management, thread management, inter-thread communication, thread synchronization, clock management, and device management. Supports extended functions including kernel-application separation, kernel-driver separation, system and application process management, system resource permission allocation, and data security isolation. - **Components:** Supports interface functions such as device driver frameworks and system maintenance management. - **POSIX Interface:** Supports unified external device interfaces, improving compatibility with different operating systems and application portability through POSIX interfaces. - **Real-Time Performance:** Kernel real-time indicators include task switching time, channel communication time, semaphore shuffling time, and signal reception time. - **Robustness:** Requires capabilities including memory fragmentation handling, memory leak countermeasures, file storage management, and permission control. - **Compatibility:** Must support normal operation on hardware platforms meeting the

resource requirements of this specification.

2.5 Application Software Requirements

Functional Requirements: - Should implement all management module functions while meeting application software environment requirements. - Implemented functions must be consistent with those explicitly described in design documents. - Should implement implicit requirements that are intrinsic, extrinsic, and directly business-related. - Contained functions should be explicit, with no hidden functions. - Should enable immediate safe stopping or safe exit. - Should handle foreseeable misoperations.

Code Safety: - Third-party or open-source code used must ensure security, with timely updates for published security vulnerabilities. - Management interfaces should be prohibited from being called by other basic application software. If external calls are necessary, the caller's access permission control should be verified.

Other requirements related to security, communication security, performance efficiency, reliability, and compatibility are omitted here for brevity.

2.6 Operating System Testing

Software Quality Testing: White-box testing is used to quickly analyze code, identify fatal defects or security vulnerabilities, and enhance operating system stability and security. Test items include control flow analysis, data flow analysis, interface analysis, expression analysis, software structure metrics analysis, semantic analysis, unit dynamic testing, software integration testing, and fault injection special testing.

Functional and Performance Testing: Tests and verifies operating system functions and performance including multithreading, multiprocessing, device access, file operations, and memory management. Test items include functional testing, real-time performance testing, and robustness testing.

Compatibility testing, driver testing, and application software testing are omitted here for brevity.

2.7 Summary of Requirements and Unspecified Technical Issues

This section summarizes the main requirements from State Grid 12180 specification for smart IoT meter management module operating systems and application software. The specification explicitly defines the numerous functions of the management module and requirements for developing functional application software. It also clearly specifies the selected operating system architecture, microcontroller requirements for the kernel, POSIX standard interfaces, and major performance and testing requirements for the operating system. Additionally,

it mandates the development of application software including basic applications, system management programs, security service programs, and extension applications.

However, the State Grid 12180 specification does not explicitly address several critical technical issues: it fails to specify unified code for management module functions, does not identify which company's general operating system product should be selected or discuss the advantages and existing problems of its application in smart IoT meters, and does not clarify requirements for process management (task scheduling), memory management, device drivers, file systems, and network systems within the selected operating system. These unspecified technical requirements must be addressed by meter enterprises after digesting and familiarizing themselves with the general operating system platform selected by State Grid, who must then determine the functional code for the management module, develop functional application software, and ultimately ensure the normal and complete operation of the management module on the operating system platform.

These unspecified technical requirements are precisely what trigger technical controversies among software experts regarding the State Grid smart IoT meter operating system.

3. Development and Application Trends of Smart IoT Meter Operating Systems: Perspectives from Before and After the Release of State Grid 12180 Specification

3.1 Development Process of State Grid 12180 Specification

The specification underwent a rigorous development process: approved for official project initiation in January 2020; establishment of the standard drafting group and initial draft creation in February 2020; supplementation of operating system and application software requirements from March to September 2020; a special meeting with meter enterprises and provincial grid experts in Fuzhou in November 2020; formation of a revised draft for comment from November to December 2020; development of a submission draft from January to April 2021; a standard review meeting from May to June 2021; formation of a final approval draft from June to July 2021; and official release and implementation in January 2022.

3.2 Initial Adoption: Yihui's Embedded Real-Time Operating System

Based on multiple information sources, the initial phase of State Grid smart IoT meters adopted Yihui's embedded real-time operating system. On June 11, 2020, at the 40th China Electrical Instrumentation Industry Development Seminar, GigaDevice (China's largest Arm MCU family and SPI NOR Flash supplier) introduced GD32 MCU high-performance general-purpose microcontrollers and their commercial applications in State Grid meters, including an

embedded real-time operating system (Yihui) based on GD32 F450. Simultaneously, when the authors inquired about the State Grid smart IoT meter operating system from Chint Group's IoT Research Institute, the institute's IoT experts provided documentation for Yihui's MS-RTOS embedded operating system.

Yihui MS-RTOS Introduction: MS-RTOS (Micro Safe RTOS) is a future-oriented IoT operating system for resource-constrained, high-security domains. Its groundbreaking feature is supporting multiprocessing and dynamic loading technologies on MCUs without MMU and with limited resources (such as Cortex-M3), enabling separate development and independent upgrading of applications and systems. MS-RTOS supports kernel space memory protection, providing high kernel security, and includes a powerful integrated development environment (IDE).

MS-RTOS Architecture: The system comprises user-state processes (Process 1-4), each equipped with memory heaps, C libraries, and C++ runtime environments; kernel-state components including a dynamic loader, I/O system, driver framework, Shell command line, and safety-critical applications; and hardware-level support. Practical functions include process management, mutexes, message queues, memory heaps, software timers, time management, interrupt management, MPU, thread management, semaphores, event flag groups, fixed-length memory blocks, condition variables, profiling modules, exception handling, and FPU. MS-RTOS fundamentally transforms traditional IoT node development methods, improves IoT node security, and meets the needs of the 5G era of interconnected everything.

Yihui Product Development Timeline: Before its establishment in 2015, Yihui developed SylixOS as a large embedded real-time operating system with complete independent intellectual property rights in China, fully substituting for similar foreign operating systems in function and performance. In 2021, Yihui launched EdgerOS, the next-generation intelligent operating system for IoT and edge computing based on the SylixOS kernel. MS-RTOS represents Yihui's operating system for resource-constrained, high-security IoT domains.

3.3 Beijing Zhixin Company: Key Player in Operating System Selection

Beijing Zhixin Company, a State Grid subsidiary specializing in power chips and operating system development, naturally plays a critical role in product selection for State Grid smart IoT meter operating systems. On August 30, 2020, the Smart Measurement Industry Alliance initiated by China Electric Power Research Institute held a technical seminar on the feasibility of a new-generation smart meter operating system in Beijing. The meeting reviewed the State Grid Metering Center's "New Generation Smart Meter Design Proposal" and Beijing Zhixin Company's "Feasibility Study Report on New Generation Smart Meter Operating System" (hereinafter referred to as the "Report"). The

seminar conducted in-depth discussions on the software architecture, kernel, and components of the new-generation smart meter operating system and provided modification suggestions for the Report.

Beijing Zhixin's Breakthroughs: On April 20, 2020, Beijing Zhixin Company's independently developed embedded operating system—Hub 4.0—passed testing and certification by the China Center for Information Industry Development. Since 2019, various edge computing core boards and products (such as transformer area fusion terminals) equipped with Beijing Zhixin's Hub operating system have been deployed in millions of units across 27 provincial grids nationwide. Subsequently, Hub 4.0 obtained functional safety system testing certification from the Ministry of Industry and Information Technology's CEPREI Laboratory, indicating that the operating system meets access conditions for high-safety standard industrial sectors such as rail transit and automotive electronics. Additionally, on April 2, 2021, State Grid Xintong Industry Group's Zhixin Company successfully passed testing for Hub Operating System 2.0 by the China Financial Certification Center. Hub Operating System 2.0 is a system-level basic platform customized for edge computing terminals, suitable for high-end complex multi-core processor devices.

3.4 Expert Discussions in the "China Modern Grid Measurement Technology" WeChat Group

Experts in the "China Modern Grid Measurement Technology" WeChat group discussed the development and application process of State Grid smart IoT meter operating systems on two occasions, presenting both positive and negative perspectives.

The first discussion on June 2, 2022, is omitted here but detailed in the authors' June 7, 2022 article "Design Defects in Domestic New-Generation Smart Meters and High-Quality Design Methods of Landis+Gyr E360 Series Smart Meters."

The second discussion on June 7, 2022, covered several key points. Experts noted that early electricity meters did use operating systems, beginning with ucos research in 2006, expanding in 2008, and generating significant debate by 2011 when many senior electricity meter experts argued that operating systems could not be applied to electricity meters. The critical question now is whether current operating systems can be applied to State Grid's 2020 standard meters.

Regarding whether operating systems reduce software development difficulty, experts argued that the purpose is to reduce product development difficulty and developer requirements without increasing costs or improving reliability—not to adopt operating systems for their own sake. The notion that operating systems are unreliable was challenged, with some noting that an operating system is essentially a task scheduler with communication mechanisms, and that open-source operating systems, having been validated, are relatively stable and reliable. However, others pointed out that while operating systems are perceived to reduce development difficulty, they actually impose high demands on

software developers, with mature ucos system developers requiring an average of one month of training to become proficient.

Experts discussed combining bare-metal reliability with multiprocessing and multitasking, though vulnerability issues may be difficult to resolve. Regarding APPs, while operating systems aim to simplify application development, special requirements related to resources, cost, and efficiency can increase usage difficulties. While a single APP may work without issues, uncertainty arises when multiple APPs are combined. For concentrator applications, adding an APP requires comprehensive testing of all APPs in its environment to ensure normal operation.

Some experts argued that a task scheduler plus communication mechanisms, industry-specific libraries, and protocol stacks essentially constitute a modern operating system, with APPs built upon libraries provided by the operating system. However, others countered that while these mechanisms and architectures are standard, actual kernel implementation is extremely complex—complexity unnecessary for electricity meters. One expert noted that TinyOS, encountered in 2005, was less than 500 bytes, while the first Linux version was only 782K (including file system), with later increases merely adding more libraries.

Experts emphasized that electricity meter-specific issues and special requirements should be incorporated into the operating system, adding electricity meter characteristic libraries to create a specialized electricity meter operating system. This requires collaboration with personnel who understand the business domain. Without such industry-specific additions, efficiency would be relatively low.

Compatibility issues between any operating system and APPs involve MCU resource allocation, particularly for low-frequency single-core embedded MCUs. The general operating system plus APP approach is not an optimal solution. For current operating systems to be used in electricity meters, the most important issue to resolve is ensuring that system resources do not change significantly or become uncertain with APP modifications or additions. For instance, stack requirements increase with additional APPs, adding system uncertainty—such issues must be properly addressed.

One expert proposed customizing high-frequency, low-cost ARM9 core chips (not exceeding 20 RMB) with built-in high-capacity flash and 64M RAM, directly running Linux systems, which would offer better compatibility and applicability than current solutions. However, concerns were raised about overall cost, power consumption (especially for vacant installations where energy efficiency ratios are problematic), and reliability, particularly regarding Linux system code running in RAM—RAM anomalies could cause system crashes, raising questions about solutions.

Regarding business models, experts debated whether operating system applications should be free, and how rights, responsibilities, and benefits should be divided. One perspective suggested offering free usage initially, gradually building a monopoly through industry chain integration before monetizing—a typical

large-scale IoT thinking and emerging trend.

3.5 MOS Embedded Operating System

During June 2-7, 2022, a metering expert asked the authors: "If we want to promote a completely self-developed operating system, what suggestions do you have?" Subsequently, the authors received an outline of the MOS embedded operating system under development.

MOS System Overview: MOS (Modular Operating System) is a modular embedded operating system derived from electricity meter bare-metal software systems. It evolves from modular software systems that developed alongside mature modular development of bare-metal software, where functional modules are completely independent and decoupled, enabling independent loading and development but requiring unified compilation of all functional modules. MOS systematizes this modular approach by introducing task scheduling and process management technologies, creating a special operating system that enables modular, APP-style software development for all electronic products without additional costs.

Key Technologies: 1. **Modular Technology:** Modules can be freely defined according to function size. For example, electricity meter measurement can be divided into measurement driver modules, instantaneous value modules, energy modules, demand modules, etc., for independent development and installation, or unified into a larger module. Each module abstracts a unified interface. 2. **Task Execution Technology:** Each module has 32 independent task messages. Once a task message is sent, the system automatically schedules the corresponding task based on priority. After execution, the task exits and waits for the next entry. Unlike traditional systems where tasks might use while(1) loops to scan or wait for new tasks without exiting, MOS requires modules to exit after task execution. This controls process numbers without adding processes or threads. 3. **Bus Technology:** All modules use a unified operation bus for data and operation access, offering high efficiency and speed without requiring multiple complex communication mechanisms found in other systems. 4. **Simple Architecture:** All functions, including drivers at various layers and application function modules, are sub-modules of the system with equal logical status, simplifying the overall architecture and increasing flexibility. Different hardware designs and functional applications can be satisfied by adding, removing, or replacing modules. 5. **Limited Preemptive Task Scheduling:** All module tasks are preemptively scheduled according to module task priority. The highest-priority processes serve the highest-priority modules, ensuring their execution is not interrupted and has limited preemption. Lower-priority task modules are executed through time-sharing preemption by lower-priority processes. This ensures real-time preemptive execution of high-priority tasks while controlling process numbers and preventing uncontrolled process creation. 6. **Multi-User Permissions:** All modules are grouped, with each group defined as a user. Modules within the same user group can access and operate each

other, while access between different user groups is prohibited. All user access permissions are controlled by security modules, and all inter-module data and operation access must be managed by the system's security module.

3.6 Author's Commentary

The current moment is opportune for discussing the development and application direction of State Grid smart IoT meter operating systems. Reviewing the 70-year development history of domestic electricity meters , the application of operating systems to implement numerous new extended functions in smart IoT meters represents a first-time achievement—a new leap in smart meter design and a novel exploration of modern computing and communication system technologies in electricity meters.

During the initial design and application phase of smart IoT meters, a multi-party cooperation model has been adopted: State Grid formulates the Q/GDW 12180-2021 "Smart IoT Meter Function and Software Specification"; Beijing Zhixin Company proposes feasibility study reports for smart IoT meter operating systems; Yihui provides the MS-RTOS embedded operating system; meter enterprises implement complete smart IoT meter design; State Grid Metering Center conducts network access testing and certification; and finally, State Grid's materials department completes unified bidding for smart IoT meter network access. This complex process necessitates establishing a smart IoT meter product quality assurance system, including a smart IoT meter operating system testing platform to evaluate meter enterprises' functional development and anti-interference capabilities.

Expert discussions on smart meter operating systems express skepticism about current general-purpose embedded operating systems and advocate for developing specialized smart IoT meter operating systems. Concerns include that general embedded operating systems have large architectures and complex kernels that complicate electricity meter design and reduce efficiency; their numerous programs lead to uncontrollable processes and stacks, RAM-triggered unreliability, and reduced overall electricity meter reliability; and that electricity meter-specific requirements should be incorporated into the operating system by adding electricity meter characteristic libraries to create specialized electricity meter operating systems. Compatibility issues between operating systems and APPs must be resolved, as the general operating system plus APP approach is not optimal.

The MOS embedded operating system proposed by metering experts, based on electricity meter bare-metal modular software systems and incorporating task scheduling, process management, bus technology, and multi-user operation technology, exhibits characteristics of a specialized electricity meter operating system, making its development approach viable.

Beijing Zhixin Company has successively developed Hub 1.0-4.0 embedded operating systems in recent years, widely applied to State Grid power equipment.

It is likely currently developing smart IoT meter operating systems and is advised to absorb domestic advanced experience in specialized electricity meter operating system development or adopt collaborative development approaches.

It is hoped that State Grid will soon organize the development of a specialized smart IoT meter operating system and explore a new journey of designing complete smart IoT meters (including metering modules, management modules, and extension modules) using embedded operating systems.

Embedded Operating System Terminology

1) Embedded Operating System (EOS): An operating system used in embedded systems, typically including low-level driver software corresponding to hardware, system kernels, device driver interfaces, communication protocols, graphical interfaces, and standardized browsers. Embedded operating systems are responsible for allocating all software and hardware resources in embedded systems, task scheduling, control, coordination, and activation of activities. They must reflect the characteristics of their host systems and achieve required functions by loading or unloading certain modules.

2) Kernel: A component of the operating system containing the system's main functions, including task scheduling, storage management, input/output (I/O), device startup, and file system management.

3) Program, Process, Thread, and Task: - **Program:** Composed of data and code. - **Process:** An executing program, consisting of the program and its execution context. - **Thread:** An execution flow of code on a process's resource platform. - **Task:** In Linux, a task refers to a process; in UCOS-II, a task refers to a thread. Tasks have three states: running, ready, and blocked.

4) Task Scheduling: Includes preemptive scheduling and non-preemptive scheduling.

References

[1] ZHAO Jian (Supervisor: FENG Quanyuan). *Design of Intelligent Home Monitoring System* [D]. Southwest Jiaotong University, 2010-05-22.

Author Biographies

ZHANG Chunhui, male (born 1938), engaged in electric energy metering technology research.

Corresponding Author: ZHANG Zhen, male (born 1977), engaged in electric energy metering technology research. Email: 721047546@qq.com

Note: Figure translations are in progress. See original paper for figures.