ISSN 2224-087X. Електроніка та інформаційні технології. 2024. Випуск 26. С. 33–45
Electronics and information technologies. 2024. Issue 26. P. 33–45

UDC 004.451.5 DOI: https://doi.org/10.30970/eli.26.4

COMPARATIVE ANALYSIS BETWEEN REAL-TIME OPERATING SYSTEMS AND SUPERCYCLES

M. Zalizovskyi, N. Huzynets

Electronic Computing Machines Department
Lviv Polytechnic National University,
12 Bandery St., UA-79013, Lviv, Ukraine
maksym.zalizovskyi.ki.2020@lpnu.ua, *nataliia.v.huzynets@lpnu.ua*

This article covers programming techniques in embedded devices, focusing on the advantages and disadvantages of real-time operating systems. With the development of technology and the growth of user requirements, approaches to programming change, and the question arises about the feasibility of using different technologies, since the new ones are not always better in certain cases. This article covers programming techniques in embedded devices, focusing on the advantages and disadvantages of real-time operating systems. With the development of technology and the growth of user requirements, approaches to programming change, and the question arises about the feasibility of using different technologies, since the new ones are not always better in certain cases. Shedding light on the intricacies of selecting the most suitable technologies amidst a backdrop of shifting paradigms.

Key words: RTOS, SuperLoop, embedded systems, architecture.

Introduction. Embedded systems are an integral part of the modern world, ensuring the functioning of a wide variety of devices — from household appliances and automotive systems to medical equipment and industrial automation. Software development for such systems requires a special approach due to strict requirements for efficiency, reliability and real-time operation. One of the popular approaches to the organization of embedded system software is the super loop architecture. Although this architecture is simple and can be effective for small, not very complex systems, it has significant limitations in the context of more complex applications.

Real-time operating systems (RTOS) solve the limitations of the superloop and other underlying architectures by providing a more precise and reliable mechanism for hardware control and task allocation. RTOS allow developers to divide complex programs into separate tasks with defined priorities and execution times, which ensures productive system operation and high performance even in the most complex applications.

The need for development in the field of RTOS is due to the growing requirements of modern technologies, such as autonomous vehicles, IoT devices and industrial automation systems. They require high reliability and accuracy, which can only be achieved with the help of advanced RTOS. Developments in this area allow creating more complex and functional embedded systems that meet the requirements of future innovative technologies.

The purpose of the study is to compare the initial and new architectural solutions, analyze the feasibility of use and compare the performance of each of the solutions.

Architecture. In embedded software development, there are several popular architectures that help organize code for efficient device operation. In this article, we will consider several of the main ones, list their advantages and disadvantages, and compare which tasks each method is best suited for.

Superloop. The Super Loop architecture is one of the simplest approaches to software development for embedded systems [1]. The main idea is to create a big endless loop in which all the functions of the program are executed in turn. Each step of the cycle is responsible for performing certain tasks, for example, processing data from sensors, performing calculations or controlling actuators.

Advantages include the simplicity of the implementation of the superloop architecture, which makes this approach attractive to beginners and those who want to quickly build a working system. A significant advantage is the high accuracy of delays, which in the execution of tasks in a sequential cycle can provide precise control over the delay time between tasks.

Another benefit is low overhead, as the lack of multiple levels of abstraction and task management means that overhead is minimal. And this, in turn, can be an advantage for systems with limited resources [2].

There are also disadvantages in this architecture. Among them - accumulation and complication of the program. This means that over time, the number of functions performed in the loop can increase, which leads to the complexity of the program and decrease in its speed.

Since all functions are executed in a single loop, it becomes difficult to manage priorities and quickly change the order of execution, resulting in a loss of flexibility. There is no parallelism in a superloop, tasks are performed sequentially, which can lead to a loss of efficiency in systems where parallelism is essential. Over time, maintenance of the application can become difficult due to the growth in size and complexity of the code, which indicates the difficulty of maintenance. Some tasks may require tighter time control than others, which may be difficult to implement within a single cycle.

Real-time operating systems. Real-time operating systems are designed to control hardware and perform tasks at a specified and predictable time [3]. These systems are specifically designed to ensure accurate and reliable performance of embedded systems where response time and accuracy are critical.

Real-time operating systems provide accurate and predictable task performance, which is critical for many embedded systems. Due to the controlled execution of tasks, the RTOS increases the reliability of the system. Such a criterion as the flexibility of the RTOS allows developers to configure the system depending on the requirements, including task prioritization. Due to the ability to skip tasks that cannot be performed at the moment, Real-time operating systems allow the system to continue working without stopping, which means continuity of operation and has a significant advantage over superloop.

Disadvantages include the following RTOS criteria. Implementation and configuration of an RTOS can require considerable effort and expertise in embedded systems programming. Commercial RTOSes can be expensive to purchase and maintain. Working with Real-time operating systems requires experience and knowledge, which can be a barrier for beginners.

When choosing between an RTOS and a superloop architecture for embedded software development, it is considered specific application examples that fit each architecture and explain why you should choose one over the other [4].

Real-time operating systems. When giving preference to real-time operating systems, it is necessary to understand the important points why you should choose them. Consider the situation when it is necessary to monitor several processes at the same time. This can be done

in several ways, one of these ways is the accumulation of microcontrollers, each of which will observe its own part. This approach is too expensive. Another method is to use one microcontroller and poll the sensors one by one, and after polling, perform calculations. In this case, a lot of time is wasted on calculations. This creates significant time gaps in which a critical situation may arise that requires an immediate solution [2]. Vital time (and such a situation is quite real) will be spent on calculations. This served as one of the reasons for the creation of real-time operating systems. They allow you to perform processes in parallel, even on single-core microprocessors, where they are mainly used. For clarity, examples of areas and advantages in which it is recommended to use RTOS are given.

Modern cars are equipped with numerous electronic systems that perform a variety of tasks, from engine management to safety systems and infotainment systems. RTOSes allow these systems to be controlled with high precision and predictability. For example, the electronic stability control (ESC) system must instantly respond to changes in vehicle driving conditions, which requires high speed data processing and control.

In the field of aviation and space, there are always requirements for high reliability and accuracy of task performance [5]. RTOS provides real-time control of the flight of aircraft or spacecraft, which is very important for the safety of passengers and crew. Medical devices such as cardiac monitors or insulin pumps must work with high precision and reliability, as the health and lives of patients depend on it. RTOSes allow for predictable and accurate control of these devices.

Superloop. Compared to the RTOS, the superloop is simple to implement and facilitates device development. It is a simple and efficient choice for less complex projects that do not require high flexibility and parallel processing, so it allows you to easily agree on a logical sequence of actions.

Examples of areas in which it is recommended to use, as well as the advantages of using superloop. In the field of home automation, for example, to control lighting or household appliances, simple embedded systems with limited capabilities are often used. The superloop architecture is suitable for such systems due to its simplicity and ease of development. Many portable devices: flashlights, watches, or small sensor devices often use superloop architecture. These devices need simple and effective solutions, which he provides. Instrument Panels and Displays: Industrial instrument panels or simple displays for displaying data often use the superloop architecture due to its simplicity and low overhead.

Working principles of architectural solutions. Architectural decisions are the foundation of any embedded system and define how the system will operate, manage resources, and perform tasks. In the section, we will consider the principles of operation of various architectural solutions used in embedded systems, including RTOS and superloop architecture [6]. These approaches have different features that affect the overall system design, efficiency, and reliability.

Superloop structure. The architecture of a superloop consists in the sequential execution of a set of actions from beginning to end in an infinite loop. This approach provides a clear structure for the execution of tasks, but the only exception is hardware interrupts, which can occur at any time. During an interrupt, the system executes a special piece of code, then returns to the current location in the loop, continuing program execution. Fig. 1 shows the implementation principle of this architecture. Tasks 1 to 4 in the Fig. 1 are the notation of individual solutions for the conditional system.

The loop continues without stopping until all the tasks in the sequence are completed, and then it starts over. However, if the application crashes or hangs during a task, it can cause the application to stop completely. There are various methods to avoid this condition, such as checkpoints, but when an error occurs, the system is forced to start over instead of repeating only the specific task [7].

This approach can be limiting in complex systems, but it has its advantages in simple applications where sequential execution of tasks is desirable or even necessary.

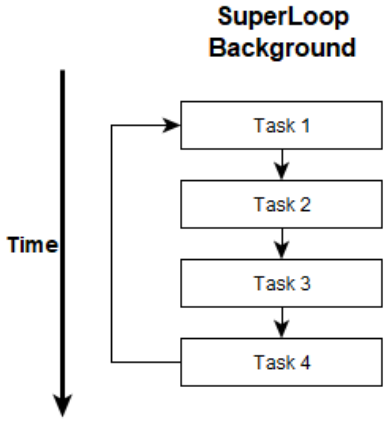

Fig. 1. Realizations of superloop architecture

The superloop architecture is often used in embedded systems, such as garden irrigation control, due to its simplicity and efficiency. The irrigation system controller is responsible for turning on and off the water valves according to the set schedule, monitoring the soil moisture level, using sensors, and displaying the system status on the display.

Consider an example of a system in which the option of using a superloop is preferable. The algorithm of the system is shown in Fig. 2. The task of this system is to measure the parameters of the environment and display the results on the screen, the diodes that signal the start and end of the measurement also work. In this case, the task is important to us that some sequence of actions is followed in a strict order and repeated for each measurement. We will describe in more detail the algorithm of the above code. We will also analyze in more detail the importance of implementing this particular architecture.

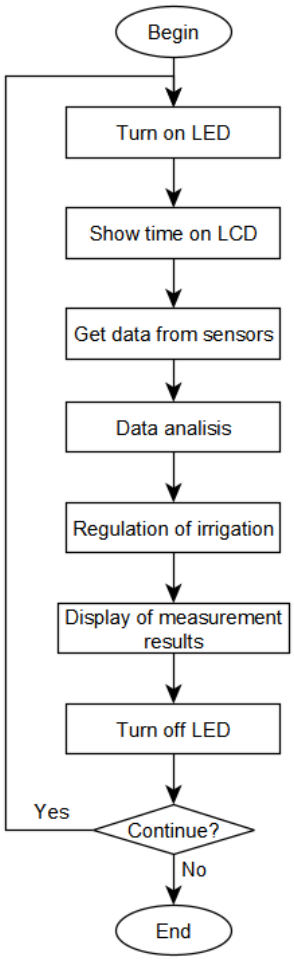

Fig. 2. Scheme of the conditional system algorithm

Consider the created block diagram for our automatic watering system. Let's assume that we have a set of parameters that must be constantly monitored: soil moisture and the amount of water that was poured into the soil. These parameters can be determined instantly and, depending on the results, either stop watering or continue. As we can see, the tasks for this system are quite primitive and do not fall under real-time requirements, since we do not have any module for exchanging parameters over the Internet or inputting parameters to which an instant reaction should occur. In addition, these tasks do not take much processor time, which could lead to a critical delay in the system. Parameters can be entered at system startup.

Firstly, turn on the LED, which signals the start of the device. The next step is to display the current time and date on the display, this is useful for tracking the execution time of various processes and for informing the user. Further, with the help of the function, the system receives the parameters from the sensors, which are required for further processing. After receiving the data, it is processed, transformed, deciphered and analyzed. Regulation of watering the lawn,

depending on the data received from the sensors. Next, the measurement results are shown on the display. The superloop ends by turning off the LED to indicate the completion of processing one cycle, i.e. one measurement. Adding a delay before starting a new cycle. This can be useful for adjusting the frequency or timing of program execution.

Using a superloop in this code has its advantages. A superloop allows you to perform a number of different functions in an infinite loop without blocking program execution. This means that the program can continuously perform different tasks without waiting for the previous ones to finish.

Also, each stage of the superloop is executed in an infinite loop, which allows the program to respond to changes during the next iteration of the loop. For example, in the given case, the program can immediately respond to the requirements of lawn watering management and display of data measured by sensors, since the complexity of implementing this system is small. In general, the use of a superloop can be useful in many cases, especially in applications that require a constant response to changes in the environment or external events.

Structure of real-time operating systems based on freeRTOS. RTOS are an important tool for developing embedded systems that require multitasking, reliability, and precise time control [8]. FreeRTOS is one of the most popular RTOS, widely used in various embedded applications. The structure of real-time operating systems based on FreeRTOS includes several key components that ensure efficient management of tasks and resources:

Tasks: FreeRTOS allows you to create multiple tasks, which are independent blocks of code that run simultaneously due to multitasking. Each task has its own priority, which determines its importance compared to other tasks. Tasks with a higher priority take priority over tasks with a lower priority.

Scheduler: The scheduler in FreeRTOS determines when and which tasks should be executed, depending on their priorities and current state. Tasks can be in different states, such as running, waiting for an event, or inactive. The scheduler selects tasks to run according to their priority and status.

Synchronization mechanisms: FreeRTOS provides various synchronization mechanisms for the interaction of tasks with each other and with other system resources. Among them are:

Semaphores: Allows you to control access to resources and synchronize the execution of tasks.

Mutexes: Provide exclusive access to resources, preventing them from being used simultaneously by multiple tasks.

Message queues: Allow tasks to exchange data or messages to coordinate their actions.

Timers: FreeRTOS supports timers that allow certain tasks to be executed at specified intervals or under specified conditions. This is useful for tasks that need to be performed periodically or in real time.

Events (Event Groups): FreeRTOS has an event group mechanism that allows tasks to respond to various events in the system. Tasks can wait for certain events and be executed after they occur.

Memory Management: FreeRTOS supports both dynamic and static memory allocation for tasks. This allows efficient use of limited memory resources, which is especially important for embedded systems.

Hardware Support: FreeRTOS supports a variety of microcontrollers and architectures, providing a wide selection of libraries and drivers for hardware interaction. This makes it easier for developers to deploy FreeRTOS in different environments.

With these core components, FreeRTOS provides developers with a powerful toolkit for building reliable, efficient, and scalable embedded systems. Ease of use and a wide range of features make FreeRTOS popular with many users in the world of embedded systems.

Users of embedded systems expect the system to work smoothly and harmoniously, performing multiple tasks at the same time. This may include, for example, instant reaction to user commands, display of current data on the screen, performance of certain operations with equipment, such as: control of sensors or switching of relays. In an ideal world, all of these actions are performed simultaneously and without noticeable delays, resulting in a smooth user experience.

FreeRTOS implements the parallel operation of the embedded system using multitasking, which allows the program to break complex processes into several independent tasks. Each task has its own priority and execution time, which ensures efficient allocation of processor time between tasks. The FreeRTOS scheduler manages task priorities and their distribution, allowing each task to run at the optimal time.

Thanks to context switching, FreeRTOS is able to quickly switch between tasks, which allows you to support parallel work and efficiently use computing resources. FreeRTOS also offers various synchronization mechanisms, such as semaphores and mutexes, which allow tasks to interact with each other and with the hardware, preventing conflicts and improving overall system performance.

Thus, FreeRTOS ensures stable and reliable operation of embedded systems, performing tasks simultaneously and in a coordinated manner. This allows users to enjoy a fast and uninterrupted system that performs all its functions effectively without noticeable delays or crashes.

Analyzing the Fig. 3 [9], it becomes clear how parallelism is realized in RTOS and perception of information by the user. First of all, each task is a certain sequence of actions that are connected to each other by means of a goal that must be achieved. Which task should be performed first and in which sequence is decided by the task manager, but the time for operations in the system is the same for everyone. With the help of such a technical solution, an illusion is created for the user that all processes are performed simultaneously, although each process takes a certain amount of time, which the user has previously set.

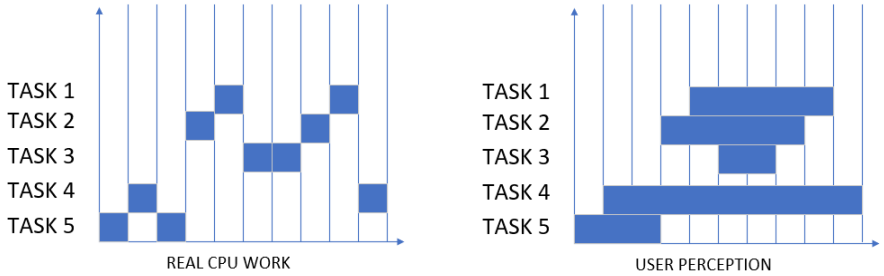

Fig. 3. Comparison of the real operation of microcontroller and user perception

Let’s consider another example of a system for which it is advisable to apply FreeRTOS. As part of the task, it is necessary to monitor the operation of a meteorological station, which analyzes parameters of humidity, pressure, temperature, direction and wind speed, and is regulated via the Internet. This system works with the following main tasks.

The thermostat regularly receives data from temperature and humidity sensors in the room, as well as, if necessary, from sensors of the external environment. This helps to determine the current state of the indoor and outdoor climate.

1) The received data is compared to the target temperature and humidity values set by the user.

2) Depending on the need, the system controls the heating or cooling of the room.

3) The system displays the current system status and all necessary settings and target values on the screen.

4) The user can change the target temperature and humidity values directly using the app on the smartphone.

Let's consider the advantages of using the FreeRTOS operating system in this system. Since this system has the potential for further expansion and improvement, it is important to provide as flexible a code structure as possible to facilitate modernization and optimization of functionality. Using the previous approach can lead to a significant increase in the amount of code and create a situation where the system does not have time to respond to user commands.

The use of FreeRTOS, on the contrary, provides convenience in the modernization and expansion of the code, and also allows you to effectively implement pseudo-parallel execution of tasks. This makes the system more dynamic and ready for further development.

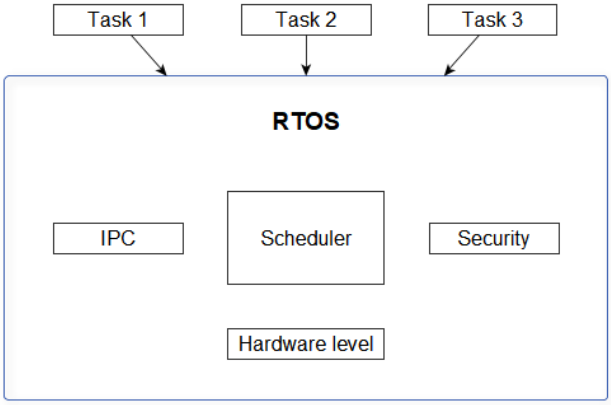

Fig. 4. General structure of FreeRTOS

The Fig. 4 [10] above shows how tasks are formed in the RTOS. First of all, all tasks are distinguished according to what they should perform. Peripheral management, security, and distribution of which tasks are entitled to execution are handled by the operating system core itself. Let's check the expediency of using RTOS based on an example with specified times for each task. Let’s take three tasks that require 30, 50 and 70 ms to complete. The context switching time will be 20 ms. Let's divide the tasks into several stages for complete completion. The first task will be completed in 2 stages, the second in 3 and the last in 4 context switches. Let’s consider in more detail. Each stage of context switching takes place in a given time interval. If we consider that we have a task that systematically monitors the change of information entered by the user in the system, then the reaction time on average will be 60 ms for three tasks. Since it takes 20ms to switch to the task and check. With a given number of tasks and a known context switching time, we can calculate how much time it takes to return to the operation we

need. When implemented by a superloop, the current time would be the time for all operations to complete.

The main problem of choosing architecture. The challenge of choosing an architecture for embedded systems is to determine the optimal approach to software organization that satisfies the specific requirements and constraints of the system. The choice of architecture determines how tasks will be executed, how the system will respond to real-time events, and how resources will be allocated between different tasks.

There are major issues and aspects to consider when choosing between Superloop and FreeRTOS. The superloop architecture is simple to implement because it involves executing tasks in a fixed loop without multitasking. This can be an advantage for small and simple systems. FreeRTOS, on the other hand, requires more complex code organization, which can require more development and debugging time.

FreeRTOS provides better response to real-time events with the ability to prioritize tasks and manage multiple threads. A superloop may not provide such a speed of response, since the tasks are executed sequentially. FreeRTOS provides more flexibility in the allocation of resources between tasks, which allows you to efficiently manage a system with different levels of task complexity. A superloop, on the other hand, limits the possibilities of simultaneous work with several tasks.

The choice of architecture can affect system power consumption. Choosing FreeRTOS can be more energy efficient due to the ability to stop unused tasks and optimize their performance. A superloop can consume more energy due to the continuous execution of the loop. If the system needs to be expanded or upgraded, FreeRTOS can provide greater scalability with its flexibility and threading capabilities. A superloop can become a limiting factor for more complex systems.

A superloop architecture can be easier to design and maintain due to its simplicity, but it can lead to complications in more complex systems. FreeRTOS may be more complex for initial development, but it can provide better long-term performance and reliability.

Ultimately, the choice of architecture for embedded systems is a matter of balancing the system requirements against the capabilities of each of the available architectures. After weighing all the pros and cons of each approach, the developer must make a decision that best meets the specific needs of the project.

Alternatives of real-time operating systems. In the world of embedded systems, there are many different RTOS that offer alternatives to typical RTOSs and may be better suited for specific tasks. Let's consider the main alternatives and their features.

Event-driven systems. Such systems focus on the processing of events that occur in the system, instead of cyclical execution of tasks. This increases efficiency because resources are only used when there is an event. Example. A system that responds to button presses or sensor events can work more efficiently using an event-driven approach.

Operating systems with optimized power use (Power-aware RTOS). In such RTOSes, special attention is paid to optimizing energy use, which can be important for battery-powered devices. Example: an environmental monitoring system that collects data from various sensors can work with such an OS to maximize the battery life of the device.

Systems with mixed criticality (Mixed-criticality Systems). Such RTOSes can provide different levels of service for tasks with different levels of criticality, optimizing performance and reliability. Example: A car's control system can have critical tasks, such as driving safety,

and less critical tasks, such as a multimedia system. Time-triggered RTOS. In such systems, tasks are performed according to a clearly defined schedule, which ensures predictable and reliable system operation. Example: Air traffic control systems can use this approach to ensure flight synchronization and safety.

Systems with a high level of security (Safety-critical RTOS). Such systems are designed with special emphasis on ensuring reliability and safety of operation, often used in industries where errors can be dangerous. Example: medical equipment or industrial process control systems can use such RTOSs to ensure safety and reliability.

Each of these alternatives has its own advantages and is suitable for specific use cases. The choice of RTOS depends on the requirements of the specific project, including needs for performance, energy efficiency, security and reliability.

Tips for choosing architecture. When developing a system, it is important to consider several key aspects to determine the optimal architecture - either a supercycle or a real-time operating system (RTOS). By anticipating the needs and characteristics of the project, an informed choice can be made that meets the requirements and resources of the system [11].

One of the key aspects is the analysis of tasks and their reaction times. If the system has tasks that require precise responses in real-time, such as process control or signal processing, an RTOS may be the better choice. On the other hand, simple tasks that do not require precise time control can be successfully performed using a supercycle.

The second aspect is the resources of the system. It is important to consider the amount of memory, processor power, and input-output capabilities. An RTOS may require more resources compared to a supercycle, so it is important to ensure that the system can handle its load.

Consideration should also be given to the complexity of development and maintenance. A supercycle may be simpler to implement and understand, which is especially useful for smaller projects or beginners. On the other hand, an RTOS can simplify the management of complex tasks and provide greater stability in meeting real-time requirements.

Finally, consider the future needs and expansion possibilities of the system. If there is expected growth in functionality or a need for greater accuracy in real-time response, it may be wiser to choose an RTOS. It is important to balance all these aspects and choose the architecture that best suits the specific needs and capabilities of the system. No option is universally better, so it is important to make an informed choice, taking into account all circumstances [12].

Conclusion. As a result of the study, an analysis and comparison of the expediency of increasing the complexity of the code was carried out, to the time advantages that each of the architectural solutions provides us.

Considering the problems of cyber-physical systems in our time, we are strongly connected with remote control over the Internet. When creating large-scale systems that must instantly respond to user actions, it is advisable to use an RTOS operating system. It enables easy upgrades and code improvements without mixing independent tasks with each other. It also gives an advantage in the speed of the system's response, since the execution of tasks is carried out pseudo-parallel, which allows us to quickly fulfill the user's request in real time.

Compared to RTOS, a superloop, which is simpler to implement, the time delay for the reaction of user actions can be significant in large-scale systems, since each task in this architecture is used sequentially, and in case of unpredictable reactions of the system, the entire execution process is stopped. Another disadvantage of the system is the difficulty of mainte-

nance. Over time, there is an accumulation of tasks performed by the system, therefore, accordingly, the volume of written code in the loop becomes significant. This makes it difficult to further modernize and improve the performance of the system. In addition to the ease of implementation, it is possible to accurately track the time at which a particular task is executed, since the task is executed from start to finish.

After comparing the timing diagrams, it becomes clear that for small tasks, RTOS can cause more harm than good and complicate the creation of a system regardless of reliability and the ability to perform tasks in parallel. However, in massive systems, the expediency of using RTOS is justified due to the ease of modernization and independence of tasks. In small tasks, the useful time to perform the operation is taken by context switching in the operating system. In the case of long-term tasks, we need to be given the opportunity to perform and calculate the next steps. Under such conditions, and especially when the speed of the system's response to the user's actions is important, it is advisable to use RTOS.

On the basis of the conducted research, we can single out the following important points for the correct choice of architecture before starting the implementation of the task. First of all, it is necessary to foresee each task that the system should perform. After receiving the list of tasks, it is necessary to set priorities and order of execution. In the case of the need for instant system feedback from the user or the execution of several tasks at the same time, it is advisable to use RTOS, because when implementing the superloop architecture, the program can give priority to the execution of the process, but in no way execute the processes at the same time. In the case of a small number of tasks that need to be performed, it is advisable to use a superloop architecture, since the tasks will be executed one after the other and will not waste time switching contexts, handing over CPU time to another task.

References

[1] *Salehi M.* Discovery and identification of memory corruption vulnerabilities on bare-metal embedded devices / M. Salehi, L. Degani, M. Roveri, D. Hughes, B. Crispo // IEEE Transactions on Dependable and Secure Computing. – 2022. – Vol. 20(2). – P. 1124-1138.

[2] *Chakrabarty A.* Exploring the Features of FreeRTOS and Testing the Performance Using Arduino - To Compare FreeRTOS Performance with ARTE / A. Chakrabarty, G. Gayathri, U. S. Manaswini, R. Maheswari // International Journal of Management, Engineering and Technology. – 2023. – Vol. 1(1). – P. 23-41.

[3] *Hee Y. H.* Embedded operating system and industrial applications: a review / Y. H. Hee, M. K. Ishak, M. S. Asaari, M. T. A. // Bulletin of Electrical Engineering and Informatics. – 2021. – Vol. 10(3). – P. 687-1700.

[4] *De Sio C.* Evaluating reliability against SEE of embedded systems: A comparison of RTOS and bare-metal approache / C. De Sio, S. Azimi, L. Sterpone // Microelectronics Reliability. – 2023. – Vol. 150. – P. 115124.

[5] *Restuccia F.* ARTe: Providing real-time multitasking to Arduino / F. Restuccia, M. Pagani, A. Mascitti, M. Barrow, M. Marinoni, A. Biondi, ... R. Kastner // Journal of Systems and Software. – 2022. – Vol. 186. – P. 111185.

[6] *Seetha R.* Benchmarking on RISC-V core and performance analysis of two open-source real-time operating systems / R. Seetha, R. Nandakumar // IEEE International Conference on Communications. – 2023. – P. 381-393.

[7] *Mamone D.* On the analysis of real-time operating system reliability in embedded systems / D. Mamone, A. Bosio, A. Savino, S. Hamdioui, M. Rebaudengo // IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT). – 2020. – P. 1-6.

[8] *Lange A. B.* HartOS-A hardware implemented RTOS for hard real-time applications / A. B. Lange, K. H. Andersen, U. P. Schultz, A. S. Sorensen // IFAC Proceedings. – 2012. – Vol. 45(7). – P. 207-213.

[9] *Mohammad A.* Real-time Operating Systems (RTOS) for embedded systems / A. Mohammad, R. Das, M. A. Islam, F. Mahjabeen // Asian Journal of Mechatronics and Electrical Engineering. – 2023. – Vol. 2(2). – P. 95-104.

[10] Mohammad, A., Das, R., Islam, M. A., & Mahjabeen, F. (2023). Real-time Operating Systems (RTOS) for Embedded Systems. Asian Journal of Mechatronics and Electrical Engineering, 2(2), 95-104.

[11] *Singh J. P.* Experimental analysis of performance paradigms for Real Time Operating System (RTOS) / J. P. Singh, S. Yadav, V. K. Chauhan, J. K. Bhatia, P. K. Singh // 11th International Conference on System Modeling & Advancement in Research Trends (SMART). – 2022. – P. 1-5.

[12] *Banbury C.* Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers / C. Banbury, C. Zhou, I. Fedorov, R. Matas, U. Thakker, D. Gope, ... P. Whatmough // Proceedings of machine learning and systems. – 2021. – Vol. 3. – P. 517-532.

[13] *Chéour R.* Microcontrollers for IoT: optimizations, computing paradigms, and future directions / R. Chéour, S. Khriji, O. Kanoun // IEEE 6th World Forum on Internet of Things (WF-IoT). – 2021. – P. 48-77.

ПОРІВНЯЛЬНИЙ АНАЛІЗ МІЖ ОПЕРАЦІЙНИМИ СИСТЕМАМИ РЕАЛЬНОГО ЧАСУ І СУПЕРЦИКЛАМИ

М. Залізовський, Н. Гузинець

кафедра електронних обчислювальних машин,
Національний університет «Львівська політехніка»,
вул. С. Бандери, 12,79013 Львів, Україна
maksym.zalizovskyi.ki.2020@lpnu.ua, nataliia.v.huzynets@lpnu.ua

У роботі проаналізовано та здійснено порівняння доцільності збільшення складності коду у контексті часових переваг, які надають різні архітектурні рішення. У зв'язку з проблемами кібер-фізичних систем сучасності, які сильно залежать від віддаленого керування через Інтернет, дослідження акцентує увагу на важливості вибору архітектури для створення великомасштабних систем, що мають миттєво реагувати на дії користувачів.

Дослідження визначає переваги використання операційної системи реального часу (RTOS), оскільки вона дозволяє здійснювати легкі оновлення та покращення коду без порушення незалежних завдань. Її псевдопаралельний механізм виконання завдань підвищує швидкість реакції системи, забезпечуючи виконання запитів користувачів у реальному часі.

У порівнянні з RTOS, архітектура суперциклу може викликати значні затримки у реакції на дії користувачів у великомасштабних системах через послідовне виконання завдань. Крім того, виникають складнощі утримання системи через накопичення завдань з часом, що ускладнює подальшу модернізацію та покращення продуктивності системи.

Після порівняльного аналізу діаграм часу стає зрозумілим, що в малих завданнях RTOS може призвести до більшого збитку, ніж користі, і ускладнити створення системи, незалежно від надійності та можливості виконання завдань паралельно. Однак у великомасштабних системах використання RTOS обґрунтоване через його зручність модернізації та незалежність завдань.

На основі проведеного дослідження виділено важливі моменти для правильного вибору архітектури перед початком впровадження певного завдання. Зокрема, модна передбачити кожне завдання, яке система повинна виконувати, встановити пріоритети та порядок їх виконання. У випадку потреби в миттєвій відповіді системи від користувача або виконання кількох завдань одночасно рекомендується використовувати RTOS. Для випадків з невеликою кількістю завдань рекомендується використовувати архітектуру суперциклу, що забезпечує виконання завдань послідовно і не втрачає час на перемикання контекстів між ними.

Ключові слова: ОСРЧ, суперцикл, вбудовані системи, архітектура.

The article was received by the editorial office on 14.05.2024.
Accepted for publication on 23.05.2024.