Hindawi
Security and Communication Networks
Volume 2020, Article ID 8871626, 9 pages
https://doi.org/10.1155/2020/8871626

Research Article

Cyborgan OS: A Lightweight Real-Time Operating System for Artificial Organ

Pan Lv, Hong Li, Jinsong Qiu, Yiqi Li, and Gang Pan

College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China

Correspondence should be addressed to Jinsong Qiu; jschiu@zju.edu.cn

Received 10 July 2020; Revised 18 September 2020; Accepted 30 September 2020; Published 14 October 2020

Academic Editor: Honghao Gao

Copyright © 2020 Pan Lv et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The software of artificial organ is more and more complex, but it lacks real-time operating system to manage and schedule its resources. In this paper, we propose a lightweight real-time operating system (RTOS) Cyborgan OS based on the SmartOSEK OS. Cyborgan OS optimizes and improves it from the code size, context switch, low power consumption, and partial dynamic update, making it suitable for the artificial organ control system. Finally, we use the heart blood pump model to analyze the task allocation and execution sequence as well as the code size of the whole program. In this application, the maximum space occupied by the code is only 15 kB, which is suitable for most microcontrollers.

1. Introduction

Cyborg is a biological creature whose functioning has been enhanced through integration of mechanical, electrical, computational, or otherwise artificial components [1]. Among them, the artificial organ and cyborg insect are typical representatives [2, 3]. From a perspective provided by Zheng et al., the human will be enhanced by artificial electromechanical components and become new cyborg intelligent systems [4, 5].

In the field of artificial organs, a lot of research work is focused on organ materials, mechanical structure design, application systems, and so on [6, 7]. Kosaka et al. have proposed a remote artificial heart monitoring system based on Internet and noninvasive sensors, which can manage the physiological state of patients and the working state of artificial heart in real time and effectively [8]. The authors in [9] propose a microcontroller pacemaker for bradycardia pacing, along with a PC-based programmer. Its algorithm is implemented in Arduino microcontrollers, which is programmed using PC and MATLAB. Markovic et al. design a blood pump for a wearable artificial kidney device. Its control system for the operation of the blood pump is composed of a central computer, power cards, and position measurement cards [10]. The power card contains a microcontroller STM32F103, which controls the power switch of the blood pump. Despite all those accomplishments, according to our limited knowledge, there is no special operating system for artificial organs. As the artificial organs are becoming more powerful, especially the application of smartphones in artificial organs, the complexity of artificial organ is greatly increased [11]. It is necessary to resort to unified management and scheduling of hardware and software resources of artificial organs. Therefore, it is advisable to adopt an embedded RTOS as the basic software platform of artificial organs. Limited by the physical conditions of human body, microcontrollers used in artificial organs generally have limited memory and very small battery capacity. This application scenario requires the embedded RTOS to be lightweight and of low power consumption at system level to extend battery life.

This paper designs and implements a lightweight real-time OS for artificial organs and names it Cyborgan OS. The main contributions of this paper are as follows:

(1) Lightweight kernel: we implement a 10 kB level kernel based on priority scheduling, which is suitable for most Microcontroller Units (MCU)

(2) Low power design: by optimizing the system operation and taking the task wake-up alignment method, we reduce the system power consumption

(3) Partial dynamic software update: to make the artificial organ have the ability of function upgrade, we have realized partial dynamic software update, which makes the software upgrade without having to halt the operating system

The rest of this paper is organized as follows. Section 2 introduces the related work. Section 3 outlines the system overview of the Cyborgan OS. Section 4 describes the basic functionality of the Cyborgan OS kernel. Section 5 presents the power management service and the partial dynamic software update service employed in implementing the Cyborgan OS. Section 6 provides a concrete application example to evaluate Cyborgan OS. Finally, the conclusions are drawn in Section 7.

2. Related Work

In the field of Internet of Things (IoT), operating system, communication, security, and cloud task scheduling are the key technologies [12–15]. Artificial organs are a typical application scenario of operating system for low-end IoT devices since artificial organs are generally built around a microcontroller with very constrained memory and small battery capacity. Hence, the same challenges are posed for OS designers for artificial organs when it comes to handling the highly limited hardware resources.

A generic OS for low-end IoT devices should have a small memory footprint, support for heterogeneous hardware, network connectivity, energy efficiency real-time capability, and security [16]. TinyOS is one of the earliest application-specific operating systems that directly deals with the limitations of sensor devices. It has a lightweight event scheduler where all program executions are performed in tasks that run to completion [17]. In contrast to TinyOS, Contiki is a lightweight operating system that supports dynamic loading and replacement of individual programs and services during run-time and without relinking [18]. Contiki has an event-driven kernel. FreeRTOS is a popular RTOS and has been ported to many MCUs [19]. Multithreading is supported by its preemptive microkernel and a tickless mode is provided for low power applications. The application firmware is written using FreeRTOS, and hence the device functionality is guaranteed and is designed to resist failures due to software malfunctions [20]. RIOT is an open-source microkernel-based RTOS that aims for a developer-friendly programming model and API [21]. Developed with the particular requirements of low-power wireless IoT devices in mind, RIOT uses an architecture inherited from FireKernel and provides support for multithreading with standard API [22].

Compared with the previous operating system, the SmartOSEK OS that our design is based on is an embedded RTOS for automotive electronics. It guarantees real-time and reliability through full preemptive task scheduling and static memory allocation [23, 24]. The kernel of SmartOSEK OS is configurable and tailorable, so its code size can be small.

In terms of embedded low-power technology, Dynamic Voltage and Frequency Scaling (DVFS) is an effective energy-saving technology, which is widely used in embedded systems. Classic DVS algorithms include the PAST algorithm proposed by Weiser, the AQRS algorithm proposed by Yao et al. [25, 26]. FreeRTOS applies a tickless idle technique to achieve more power saving during the processor idle periods [27].

For the moment, most dynamic software updates designed for embedded systems are achieved via bootloader, which is a part of the microcontroller [28–30]. But the software update method via bootloader has a drawback that the update involves the whole system and will halt the entire system when the update is underway. When it comes to artificial organs, as they are already implanted in the human body, halting the control system can be dangerous and is catastrophic. Also it is mentioned in [31] that the communication rate of artificial organs is relatively low, and the partial dynamic software update method can reduce the amount of code to be updated and reduce the update time consumption. In this study, we propose a method of partial dynamic software updating, which can update the program without stopping the system.

3. System Overview

The Cyborgan OS consists of kernel, service, and driver as shown in Figure 1. In the OS kernel, task, interrupt, and alarm are the minimum function subsets to realize the operating system functions. The other four modules can be tailored according to the needs. In this way, the memory footprint of the OS kernel can be reduced as much as possible, so as to adapt to more types of microcontrollers.

The service layer abstracts common functions into services and provides them for system or application calls. At present, we have implemented two services: power management service and partial software update service. The driver layer realizes the basic hardware drivers such as microcontroller drivers, communication drivers, and I/O drivers. Given that drivers are already very common technologies, this paper will not introduce them in detail. As mentioned above, the Cyborgan OS is designed to provide a lightweight operating system kernel with low power consumption to provide system level support for the complex functionality of artificial organs. Therefore, its design should follow several principles:

(1) Real-time: In common scenarios of real-time control systems such as blood pump of artificial heart, it is often required that the microcontroller gives a response within a predefined time interval. Hence, the Cyborgan OS should support real-time task scheduling.

(2) Tailorability: The Cyborgan OS needs good tailorability, because when constructing different applications, irrelevant modules can be deleted considering the limited memory resources of the microcontroller chip.

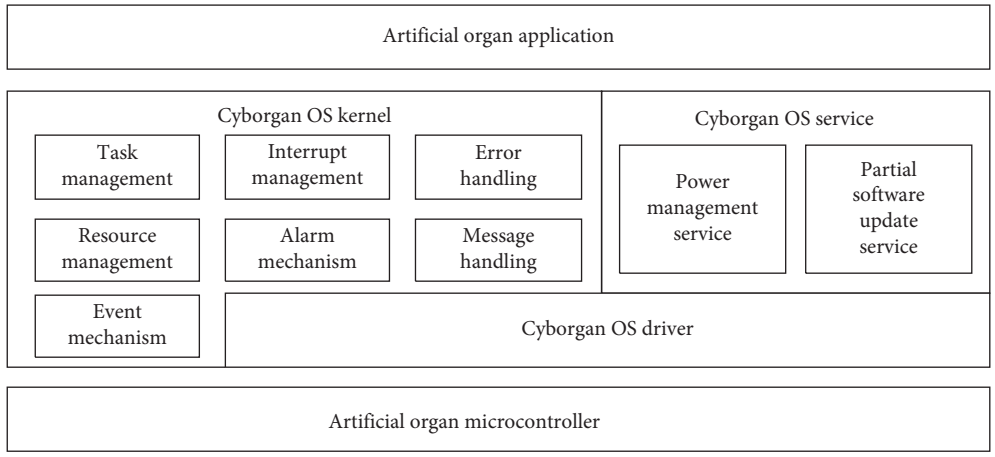

Figure 1: Architecture of Cyborgan OS.

(3) Low power design: Artificial organs installed inside human bodies often cannot obtain stable and continuous power supply. Hence, the operating system should have low power consumption and can switch dynamically between operating modes in accordance with the task status.

(4) Partial dynamic software update: During the software update, the normal functioning of artificial organs cannot be halted, and the update should be done as quickly as possible. So partial dynamic software update is applied to upgrade the functions of artificial organs.

4. Cyborgan OS Kernel

The kernel of Cyborgan OS initially evolved from the SmartOSEK OS [24]. It provides basic functions such as task management, interrupt management, alarm mechanism, resource management, event mechanism, messages handling mechanism, and error handling mechanism. The last four functions can be tailored dynamically according to the requirements of artificial organ applications to reduce the kernel size. All other functional components, such as power management and partial dynamic software update, are kept separate from the kernel and described in the next section.

4.1. Task Management. In common scenarios of real-time control systems such as the blood pump of artificial heart, the control software of artificial organs can be subdivided into functions implemented in the form of tasks. It is often required that the microcontroller give a response within a predefined time interval. Hence, the Cyborgan OS supports fully preemptive scheduling policies. For example, in the application of artificial heart, blood pump must run periodically, its control task should have the highest priority, and it can preempt low priority tasks such as external communication.

In order to save the task switching time of Cyborgan OS, we optimize the task switching according to the method in paper [32]. As shown in Figure 2, the basic idea is to divide

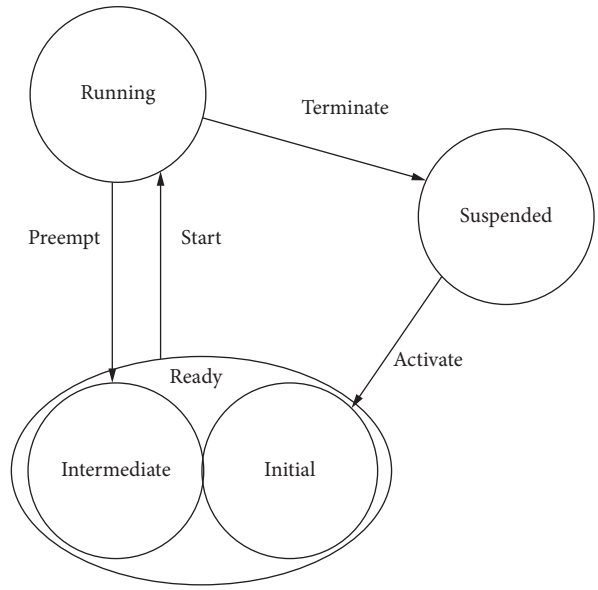

Figure 2: The improved basic task model [32].

the ready state into intermediate state and initial state. According to different states, different task context switching methods are used to improve the task switching speed. When the current running task is terminated into suspended state, it is not necessary to save the context because the task in the suspended state has finished. The task in initial state that is ready to run is a completely new task, so there is no context to recover from. In other words, context switching is needed only when the task enters or leaves the intermediate state.

In the application of artificial organs, the execution of tasks is often serialized, and preemption of normal tasks rarely occurs; that is, tasks rarely enter the intermediate state. Therefore, this context switching method can improve the speed of task switching.

4.2. Messages Handling Mechanism. The Cyborgan OS uses OSEK COM mechanism to realize message handling. OSEK COM supports m:n-communication [33]. It supports two

message types: internal message and external message. The internal message refers to the message transmission between tasks in MCU, which is sent and received using shared memory. The external message is used between MCU through an external physical bus such as SPI.

As shown in Figure 3, when sending an external message, OSEK COM packs the message in protocol data unit (PDU) and then sends it through the transmission layer. Receiving messages is a reverse process. In this way, the OS can shield the difference of communication hardware and achieve good portability.

4.3. Alarm Mechanism. Based on counters, the Cyborgan OS offers alarm mechanisms to handle periodic tasks such as sensor acquisition. A counter tracks the number of ticks and at each tick, the alarm value gets incremented by one. When the alarm value equals the predefined counter value, the OS will activate a task, set an event, or call an alarm callback routine, as shown in Figure 4. More than one alarm can be attached to the same counter. In addition, the OS provides services to cancel alarms and to get the current state of an alarm.

In the application of artificial organs, there are often periodic tasks such as sensor acquisition and actuator control. In this case, the alarm mechanism can be used to activate the corresponding tasks regularly.

4.4. Interrupt Management. The OS encapsulates the interrupt handling mechanism, and the ISR is subdivided into two categories:

(1) ISR category 1: The ISR category 1 does not call any system service. Once the ISR is finished, the systems return to the instruction where the interrupt has occurred. This ISR category does not affect task management.

(2) ISR category 2: The user routine in the ISR category 2 can call system services. When the ISR is done and a preemptable task is interrupted, rescheduling will take place.

In Cyborgan OS, interrupts are handled by hardware and can interrupt tasks. If a task is activated from an interrupt routine, the task will be scheduled at the end of all running or queued interrupt routines.

4.5. Resource Management. The resource management module handles the situation when tasks of different priority visit simultaneously the shared resources such as the memory and hardware devices. When multiple tasks access the same resource, because the low priority task occupies the resource first, the high priority task sharing access to the resource has to wait the resource; it then makes other low priority tasks run before these high priority tasks. To avoid priority inversion, the priority ceiling protocol is used in the resource management module.

In this protocol, each resource is assigned a priority ceiling, which is a priority higher than or equal to the highest priority of any task, which may lock the resource. When a task locks a resource, its priority will be raised immediately to the priority ceiling of the resource. When the task releases the resource, its priority will be dynamically reset to its default priority. A task can lock several resources and is required to release them in the Last In First Out (LIFO) order.

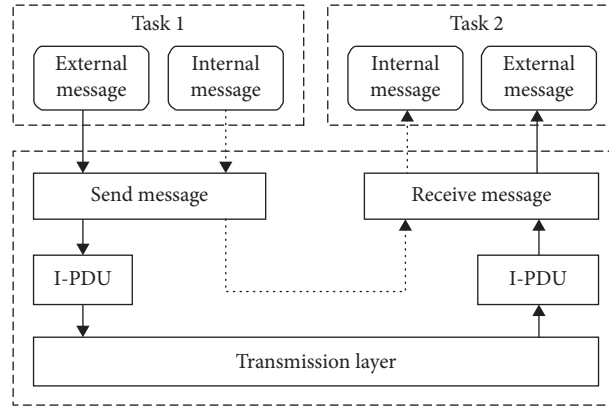

Figure 3: Structure of message handling mechanism.

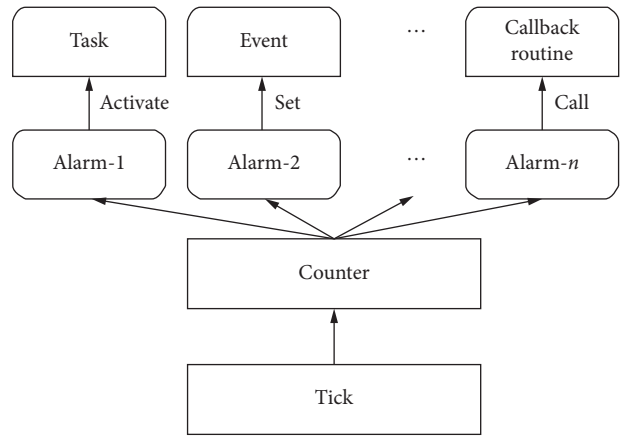

Figure 4: Structure of alarm mechanism.

4.6. Event Mechanism. The event mechanism is used for task synchronization and is provided only for extended tasks to help these tasks to switch between the wait state and other states. In the system configuration, each extended task is assigned one or several events and is called the owner of these events. An event can only be removed by its owner, and only its owner will wait for it to occur. The OS provides interfaces to set, remove, or wait for events. If a running extended task requires an event that has not yet occurred, the task will switch into the wait state. If the event that a blocked extended task is waiting for occurs, the task will enter the intermediate state. And if the event that a running extended task is waiting for has already occurred, the task will continue running.

4.7. Error Handling Mechanism. Besides handling errors by looking at the returned values, the OS also provides a hook routines mechanism in which users can define how to handle

certain errors in hook routines. The priority of error handling function is higher than that of any task so that errors can be handled on time. For example, when a sensor of an artificial organ fails, the application can call hook routines to deal with the fault ahead of all other tasks.

5. Cyborgan OS Service

In order to better meet the needs of artificial organs for low power consumption and program update, we extend the two services of power management and partial dynamic software update based on the operating system kernel. These services can be started or shut down by the application.

5.1. Power Management Service. Based on the low power modes of the microcontroller, Cyborgan OS implements a power management service that enables the operating system to switch between run mode, sleep mode, and stop mode and takes the task wake-up alignment to further alleviate the impact of task switch on system power consumption [34].

5.1.1. System Operation Optimization. In an embedded RTOS, a task is generally activated by event-based triggers or time-based triggers. In the event-based triggering, a task is activated when corresponding event such as an interrupt takes place. After the task is completed, the operating system performs the idle task and waits for the next event to trigger another task. The time-based triggering makes use of the alarm mechanism to periodically activate certain tasks. When the task is executed, the operating system will again perform the idle task. In the SmartOSEK OS, the idle task will do nothing except incrementing an uint8 variable in an infinite loop, which is a waste of system resources.

To avoid this waste, the Cyborgan OS will automatically switch to the sleep mode when the idle task is activated. When it needs to terminate the idle task and to perform other tasks, it will automatically switch to the run mode. Assuming that the idle task occupies 50% of the CPU time and the current in the run mode and in the sleep mode is 4 mA and 0.1 mA, respectively, after taking this mode-switching method, the average current is reduced from 4 mA to 2.05 mA, and the power consumption drops by 48.8%, which are significant advances.

5.1.2. Task Wake-up Alignment. In the implementation of the Cyborgan OS, the system tick is triggered by the alarm interrupt and is triggered every 1 ms in the default setting. When the system enters the sleep mode, even if there is no user interrupts tasks to be handled, the system tick will make the operating system switch from the sleep mode to the run mode every 1 ms. Figure 5(a) shows the interference from the system tick with the system operation. In [27] tickless idle technique is implemented in FreeRTOS to solve this problem. As shown in Figure 5(b), the tickless idle implies disabling the tick sourced from the idle task function, in order to ensure that the processor remains in the low power mode for longer periods of time. At the moment of entering the low power mode, the information of when a user task has to continue execution is obtained from the delayed task list and is used for the configuring of the timer to generate interrupt and wake up processor just before the task has to be deblocked.

(1) No optimization will bring unnecessary interruptions caused by the system tick, since the system tick changes the value of the timer by the interrupt. Even if there is no user task, it will leave the sleep mode due to the system tick interrupt. Taking this strategy, the power consumption (P_c) is

$$P_c = (t1 + t2 + t3 + t\text{Tick} \times \text{tickCnt}) \times a1 + t\text{Sleep} \times a2 + (\text{tickCnt} + 3) \times 2 \times a3. \quad (1)$$

(2) Applying the tickless idle technique, the system tick is temporarily disabled when the system enters the sleep mode and a general-purpose timer clocked with the system clock is used. So unnecessary system tick interrupts will not be generated in the sleep mode, and the power consumption is

$$P_c = (t1 + t2 + t3) \times a1 + t\text{Sleep} \times a2 + 3 \times 2 \times a3. \quad (2)$$

(3) On the basis of the tickless idle technique, the task wake-up alignment further reduces the number of switches between the run mode and the sleep mode. The power consumption is

$$P_c = (t1 + t2 + t3) \times a1 + t\text{Sleep} \times a2 + 2 \times a3. \quad (3)$$

In the scenario of artificial organs, most tasks are periodic, so it suffices to trigger them periodically. For this reason, this paper proposes the task wake-up alignment to optimize the system. The task wake-up alignment implies waking up for execution at the same time those tasks need to be waken up. As shown in Figure 5(c), when the period of one task is multiple that of another, we can select a time interval and wake up the tasks periodically. After each wake-up, the waken up tasks are triggered and executed in batch. After completing these tasks, the operating system performs the idle task and enters the sleep mode. Compared with the tickless idle technique, the wake-up alignment can further reduce the number of wake-ups and hence the power consumption.

As shown in Figure 5, Task1, Task2, and Task3 are needed to be waken up during period T. $t1$, $t2$ and $t3$ represent the execution time of Task1, Task2, and Task3, respectively. tSleep represents the time the system is in sleep mode. tTick represents the execution time of each system tick. tickCnt represents the number of ticks in period T. It is assumed that the current is $a1$ in the run mode, $a2$ in the sleep mode, and $a3$ during the mode switch. The three different strategies are analyzed in detail as follows:

From the above analysis, we conclude that the task wake-up alignment can achieve lower power consumption.

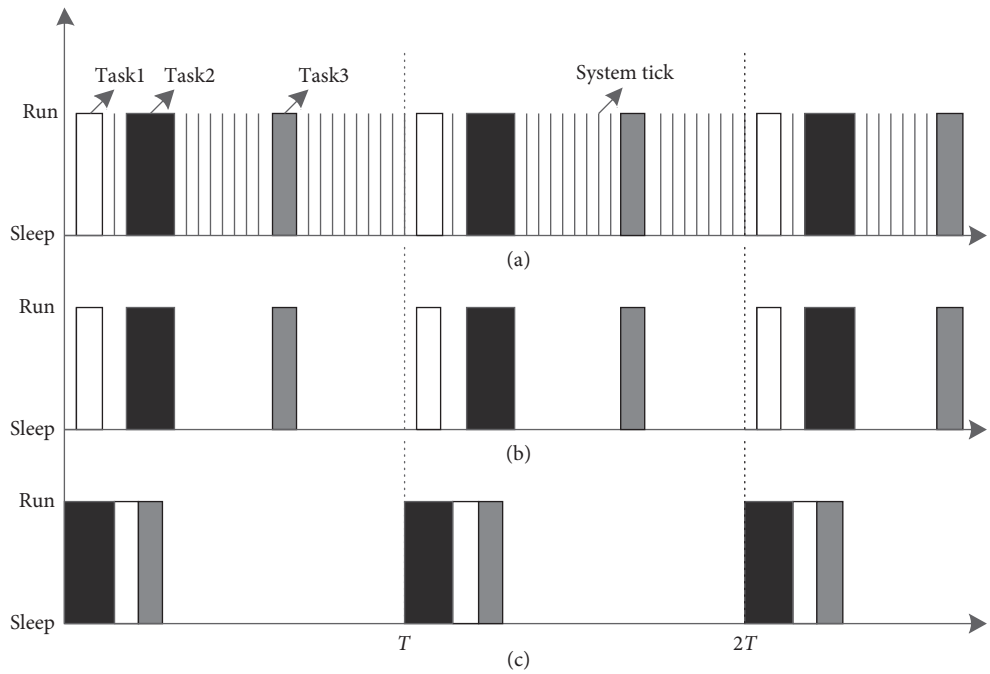

Figure 5: Operating system run/sleep mode switch using three strategies: (a) no optimization, (b) tickless idle technique, and (c) task wake-up alignment.

5.2. Partial Dynamic Software Update Service. The application scenario of artificial organs poses two challenges for designers of software update: (1) how to update the program without affecting the system's normal functioning? (2) If the update fails, how to do the version rollback of the program? To solve these two questions, this paper proposes a partial dynamic software update service.

The key of partial dynamic software update lies in the incremental update and rollback mechanism, which means that every single update targets only one function, burning in the new version of the function to replace the old one. Each function that needs to be updated is placed in a predefined program section, ensuring that the update is performed at the function level.

Partial dynamic software update divides the microcontroller's memory into a fixed code area and updatable code area, as shown in Figure 6. The kernel of the operating system resides in the fixed code area since it is not allowed to modify the kernel of the operating system. In the early stage of designing the application software for the artificial organs, it is necessary to estimate which functions may need to update afterward and place these functions in the updatable code area.

The function is the minimum component that can be updated by the dynamic software update in part. Taking advantage of the compiler's ability to assign the compiled code to a designated address, we put each function that needs update separately in a self-defined program section. The self-defined program section is physically mapped to the updatable code area. According to the estimated number of functions that may need to be updated, we partition the updatable code area to make it convenient to store the updated program.

Once the patch file is received by the microcontroller of artificial organs via Wireless Body Area Networks (WBAN) [35], it will be parsed and safety-checked. After the program is burnt into the designated memory, the corresponding function pointer will be modified so that the application program can use the newly updated function. The framework of the dynamic software update in part is shown in Figure 7.

We designed a rollback mechanism to improve reliability. When the data verification or program burning fails, the update will be stopped and the previous version of the program will be reloaded so that the program continues to run the old version of the program.

6. The Application of Cyborgan OS

In this paper, we instantiate the operating system based on NXP KL02 microcontroller, which only has 32 kB ROM and 4 kB RAM. To verify the feasibility of the Cyborgan OS for artificial organs, this paper tests it in a heart-blood pump model. The heart-blood pump model uses an artificial blood pump to replace the ventricle. The model can perform respiratory monitoring, temperature monitoring and blood pump control, etc. When the patient's respiratory rate changes, the blood pump output is controlled in real-time. Closed-loop control is realized in this heart-blood pump model. Figure 8 is a picture of the heart-blood pump model.

6.1. Hardware Framework. In the heart-blood pump model, the blood pump control node (KL02-A) and the sensor acquisition node (KL02-B) are responsible for the control and data acquisition of all the peripherals. Both nodes run

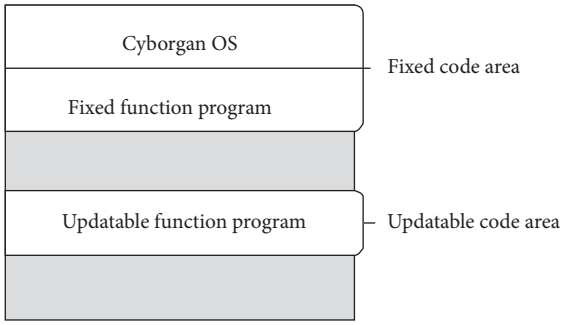

Figure 6: Layout of memory.

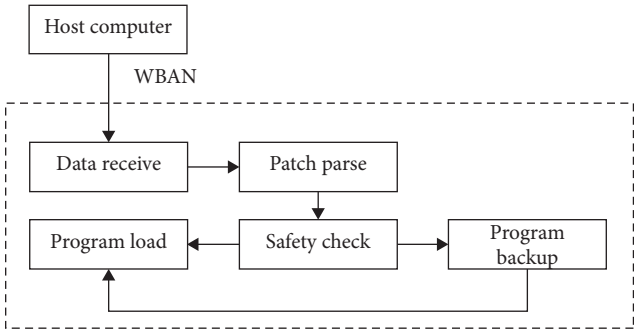

Figure 7: Framework of the partial dynamic software update.

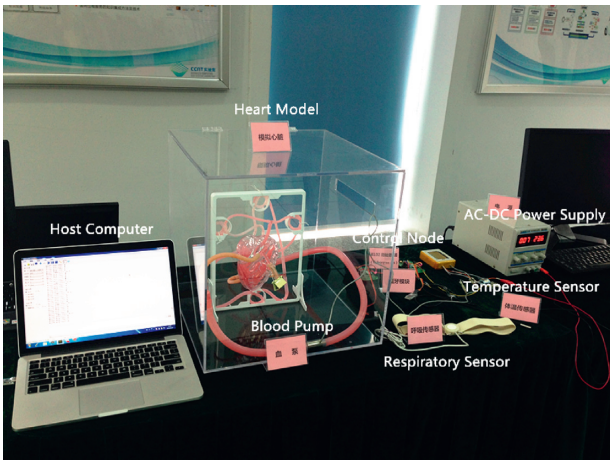

Figure 8: Heart-blood pump model.

Cyborgan OS and communicate with each other by the I^2C bus. The hardware framework is shown in Figure 9.

6.2. Application Software Design. According to the improved method mentioned above, we optimize the task allocation, make it run serially, and reduce the time of context switching. As shown in Figure 10, task OsTaskRecv and OsTaskCtrl run on KL02-A, in which OsTaskRecv receives the sensor data sent from KL02-B and OsTaskCtrl controls the blood pump. They are activated in a sequence of 100 ms cycles. Three tasks named OsTaskTemp, OsTaskRate, and OsTaskSend run on KL02-B, which are responsible for body temperature, respiratory rate acquisition, and sensor data transmission to KL02-A. According to the task wake-up alignment strategy, the OS activates these three tasks in sequence every 100 ms too.

When the blood pump is in normal operation, it pumps about 73 times/minute. If the respiratory rate or body temperature is too high, the blood pump control state will switch into an emergency state and gradually increase its pumping frequency. When exiting from the emergency state, the blood pump will gradually lower its pumping frequency in the same way, finally returning to 73 times/minute.

We used the CodeWarrior development environment to integrate the application code running on KL02-A and KL02-B with the Cyborgan OS source files into a project.

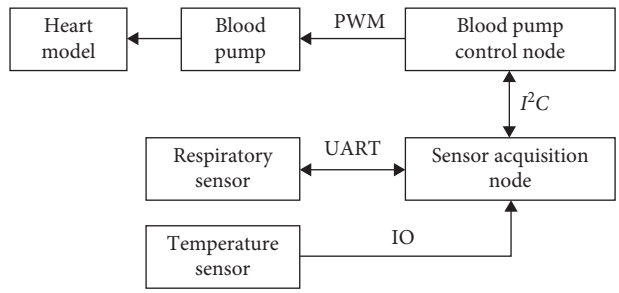

Figure 9: Hardware framework of the heart-blood pump model.

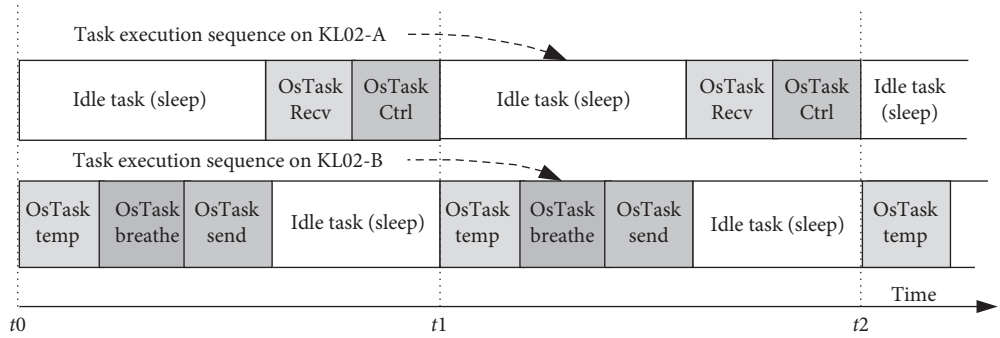

Figure 10: Task execution sequence on two nodes.

Table 1: Sizes of data segments of the ELF files generated in the heart-blood pump model.

Segment	KL02-A (byte)	KL02-B (byte)
.text	13240	11040
.data	56	66
.bss	1932	2508

Then we compile and link this project to generate the object code ELF files. Finally, the ELF file is parsed with the greadelf tool to get the sizes of data segments, as shown in Table 1.

As can be seen from Table 1, Cyborgan OS is lightweight enough to run on the KL02 platform with very limited resources.

7. Conclusion

With the increasing complexity of artificial organs, embedded real-time operating system as its basic software platform becomes inevitable. However, due to the limitations of the physical space of artificial organs, hardware resources are limited. Therefore, code size, low power consumption, and partial dynamic software update have become the three key points of artificial organ operating systems. According to the application scenario of artificial organ, this paper puts forward the design principle and structure of artificial organ operating system. According to these principles, Cyborgan OS reduces the code size of the kernel without sacrificing the basic performance, implements a low-power strategy, and adopts the partial dynamic software update method. The operating system has the characteristics of real-time, low power consumption, and small code space, which will be further extended to the other fields of Internet of things and edge computing in the future.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

Part of the paper has been published in CPSCom-2019. The work has been supported by the National Natural Science Foundation of China NSFC61602404, the National Key Research and Development Program of China 2017YFB1301102, and National Important Science & Technology Specific Projects 2017ZX01038201.

References

[1] D. Mertz, *The International Encyclopedia of Communication*, International Communication Association (ICA), Washington, DC, USA, 2008.

[2] N. Zheng, Q. Ma, M. Jin et al., "Abdominal-waving control of tethered bumblebees based on sarsa with transformed reward," *IEEE Transactions on Cybernetics*, vol. 49, no. 8, pp. 3064–3073, 2019.

[3] H. Hong, X. Wang, Z. Zhu, Q. Ma, N. Guan, and N. Zheng, "An HSR data model for cyborg insect research experiments," *Chinese Journal of Electronics*, vol. 27, no. 4, pp. 680–686, 2018.

[4] N. Zheng, L. Su, D. Zhang, L. Gao, M. Yao, and Z. Wu, "A computational model for ratbot locomotion bsed on cyborg intelligence," *Neurocomputing*, vol. 170, pp. 92–97, 2015.

[5] Z. Wu, N. Zheng, S. Zhang, X. Zheng, L. Gao, and L. Su, "Maze learning by a hybrid brain-computer system," *Scientific Reports*, vol. 6, no. 1, Article ID 31746, 2016.

[6] N. H. Cohrs, A. Petrou, M. Loepfe et al., "A soft total artificial heart-first concept evaluation on a hybrid mock circulation," *Artificial Organs*, vol. 41, no. 10, pp. 948–958, 2017.

[7] C. Fox, S. Chopski, N. Murad et al., "Hybrid continuous-flow total artificial heart," *Artificial Organs*, vol. 42, no. 5, pp. 500–509, 2018.

[8] R. Kosaka, Y. Sankai, R. Takiya, T. Jikuya, T. Yamane, and T. Tsutsui, "Tsukuba remote monitoring system for continuous-flow artificial heart," *Artificial Organs*, vol. 27, no. 10, pp. 897–906, 2003.

[9] M. Sayahkarajy, E. Supriyanto, M. H. Satria et al., "Design of a microcontroller-based artificial pacemaker: an internal pacing device," in *Proceedings of the 2017 International Conference on Robotics, Automation and Sciences*, pp. 1–5, Melaka, Malaysia, November 2017.

[10] M. Markovic, M. Rapin, M. Correvon, and Y. Perriard, "Design and optimization of a blood pump for a wearable artificial kidney device," *IEEE Transactions on Industry Applications*, vol. 49, no. 5, pp. 2053–2060, 2013.

[11] N. Reiss, T. Schmidt, M. Boeckelmann et al., "Telemonitoring of left-ventricular assist device patients-current status and future challenges," *Journal of Thoracic Disease*, vol. 10, no. 15, pp. S1794–S1801, 2018.

[12] H. Gao, C. Liu, Y. Li, and X. Yang, "V2VR: reliable hybrid-network-oriented V2V data transmission and routing considering RSUs and connectivity probability," *IEEE Transactions on Intelligent Transportation Systems*, vol. 99, 2020.

[13] H. Gao, Y. Xu, Y. W. Yin, R. Li, and X. Wang, "Context-aware QoS prediction with neural collaborative filtering for Internet-of-Things services," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4532–4542, 2020.

[14] X. Zhang, H. Gao, H. Xu, and M. Bian, "An IoT-based task scheduling optimization scheme considering the deadline and cost-aware scientific workflow for cloud computing," *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, no. 1, 19 pages, 2019.

[15] Y. Xu, L. Li, H. Gao, L. Hei, R. Li, and Y. Wang, "Sentiment classification with adversarial learning and attention mechanism," *Computational Intelligence*, 2020.

[16] O. Hahm, E. Baccelli, H. Petersen, and N. Tsiftes, "Operating systems for low-end devices in the internet of things: a survey," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 720–734, 2016.

[17] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," *ACM SIGARCH Computer Architecture News*, vol. 28, no. 5, pp. 93–104, 2000.

[18] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki-a lightweight and fiexible operating system for tiny networked sensors," in *Proceedings of the 9th Annual IEEE International Conference on Local Computer Networks*, pp. 455–462, Tampa, FL, USA, November 2004.

[19] R. Barry, *FreeRTOS: A Free Open Source RTOS for Small Embedded Real Time Systems*, Elsevier, Amsterdam, Netherlands, 2003.

[20] V. J. Devi, "Artificial cardiac pacemaker," in *Proceedings of the 2017 Third International Conference on Science Technology Engineering & Management*, pp. 1015–1017, New York, USA,, March 2017.

[21] E. Baccelli, C. Gundogan, O. Hahm et al., "RIOT: an open source operating system for low-end embedded devices in the IoT," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4428–4440, 2018.

[22] H. Will, K. Schleiser, and J. Schiller, "A real-time kernel for wireless sensor networks employed in rescue scenarios," in *Proceedings of the 2009 IEEE 34th Conference on Local Computer Networks*, pp. 834–841, Berlin, Germany, November 2009.

[23] OSEK Group, "OSEK/VDX operating system specification 2.2.3," 2003, http://www.osek-vdx.org.

[24] L. V. Hong, *Design and Implementation of SmartOSEK OS 4.0 Consulting AUTOSAR*, Zhejiang University, Zhejiang, China, 2010.

[25] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pp. 374–382, Milwaukee, WI, USA, October 1995.

[26] L. Chandrasena, P. Chandrasena, and M. Liebelt, "An energy efficient rate selection algorithm for voltage quantized dynamic voltage scaling," in *Proceedings of the 14th International Symposium on Systems Synthesis*, pp. 124–129, Montréal, Canada, October 2001.

[27] M. Simonovic and L. Saranovac, "Power management implementation in freeRTOS on LM3S3748," *Serbian Journal of Electrical Engineering*, vol. 10, no. 1, pp. 199–208, 2013.

[28] W. Yang and L. Luo, "The dynamic loading of the module in the embedded system," *Microcontroller & Embedded System*, vol. 11, pp. 8–10, 2005.

[29] M. Wahler, S. Richter, and M. Oriol, "Dynamic software updates for real-time systems," in *Proceedings of the 2nd International Workshop on Hot Topics in Software Upgrades*, New York City, NY, USA, January 2009.

[30] H. Wang, T. Wang, Q. Wang et al., "Remote update mechanism of reliable embedded Software based on bootloader," *Microcomputer Information*, vol. 23, no. 7, pp. 57–59, 2007.

[31] M. N. Islam and M. R. Yuce, "Review of medical implant communication system (MICS) band and network," *ICT Express*, vol. 2, no. 4, pp. 188–194, 2016.

[32] Z. Wu, H. Li, G. Yang, Z. Gao, and P. Lv, "An improved method of task context switching in OSEK operating system," *International Journal of Pervasive Computing and Communications*, vol. 6, no. 2, pp. 179–191, 2010.

[33] OSEK Group, "OSEK/VDX communication specification 3.0.1," 2003, http://www.osek-vdx.org.

[34] P. Lv, Y. Li, H. Li et al., "Optimization methods of operating system for artificial organs," in *Proceedings of the 2019 IEEE Cyber, Physical and Social Computing*, pp. 222–228, Atlanta, GA, USA, July 2019.

[35] M. Usman, M. R. Asghar, I. S. Ansari, and M. Qaraqe, "Security in wireless body area networks: from in-body to off-body communications," *IEEE Access*, vol. 6, pp. 58064–58074, 2018.