APPLICATION OF FreeRTOS FOR IMPLEMENTATION OF THE EXECUTION ENVIRONMENT OF REAL-TIME MULTI-VERSION SOFTWARE [1]

Mikhail V. Saramud, Igor V. Kovalev, Vasiliy V. Losev, Mariam O. Petrosyan

Reshetnev Siberian State University of Science and Technology, Krasnoyarsk
e-mails: msaramud@gmail.com, basilos@mail.ru
Russian Federation

Abstract: The paper describes the creation of a unified means for developing components of cross-platform on-board software in the form of an integrated development environment (IDE). The technical features of creating a multi-version real-time execution environment based on the real-time operating system (RTOS) are described. The environment is an embedded real-time control system. The analysis of the existing RTOS was carried out, FreeRTOS was chosen as the basis for system under development. The technical features of the modifications necessary to ensure the required functionality of the system are described. Examples for the decision block and executable versions are described. The practical implementation of the proposed solutions, which confirms their applicability and effectiveness, is considered.

Key words: N-version programming; fault tolerance; reliability; real-time operating system; execution environment.

1. INTRODUCTION

Currently industries that require reliable, fault-tolerant real-time management systems are actively developing. These include high-tech industries that use composite and dangerous materials, autonomous unmanned objects - from multi-rotor systems to vehicles with autopilot function and motorized seats with voice control for people with disabilities.

Software is an integral part of modern control systems; however, only simple software can be created without errors with guarantee [1]. Modern control software is used to solve increasingly more complex tasks, increase in the volume of processed information is avalanche-like. With the increasing complexity of software, the likelihood of errors in it grows.

[1] This work was supported by Ministry of Education and Science of Russian Federation within limits of state contract № 2.2867.2017/4.6

Traditionally, methods for reliability of embedded control systems (CS) improvement have been based on increasing the reliability of hardware. For this exist classical tools and approaches that provide the required level of reliable operation of the CS, including the use of duplication of CS components, which makes no sense for the software, since in this case, program errors will be duplicated [2].

At present, the development and application of software fault tolerance methods is urgent, which is associated with the complication and enlargement of software systems, an increase in the number of functional CS modules, which leads to an increase in the probability of system failure due to the failure of one of the modules [3]. The growth of the reliability of modern hardware CS is provided due to the development of technological processes for the production of radio electronic equipment, and due to cost reduction, duplication of hardware components is simplified.

Embedded real-time control systems are used mainly on devices for which the possibility of autonomous operation is critical. Nowadays, the most popular representatives of this class are unmanned aerial objects (UAV). This class includes multi-rotor systems and winged drones, which are actively used in practice.

2. ANALYSIS OF THE TARGET PLATFORM FOR ONBOARD SOFTWARE

As the object we are considering UAV, its control system is on-board software. Since this is a rather heterogeneous class of devices, it is necessary to create cross-platform, for the possibility of applying our solution on the widest possible range of devices [4]. Thus, our task is to develop cross-platform on-board software based on the multi-version technology of software redundancy in the real-time execution environment.

Especially critical is the reliability of software in real-time systems. If in regular system in case of failure, there is time to process it, restore the state of the system before failure and repeat the calculations, in case of real-time operation there is no possibility of restoring and re-executing the task [5], and after the execution time expires the task will be completed forcibly.

The most rational way to create a multi-version real-time execution environment is to take as a basis the existing real-time operating system (RTOS), with open original source code and the possibility of its modification and use for your own needs, including commercial ones. The independent development of the RTOS from scratch, its porting to a variety of architectures, debugging and testing, further development of the system will require very large material and time costs, without providing significant advantages over the use of existing systems [6].

Consider the existing RTOS:

FreeRTOS - one of the most popular RTOS today. Ported to a huge number of hardware platforms.

Advantages: free of charge, ported to a large number of hardware platforms, powerful functional, various libraries: from graphics to network protocols, the operating system is well documented.

Disadvantages: the complex process of porting to new hardware platforms, however, at the current moment there are many ports to all common platforms and the community is developing ports for newly emerging ones.

KeilRTX - until recently, this RTOS was commercial, but it has recently become open. Works only on ARM architecture.

Advantages: free of charge, easily ported to a large number of hardware platforms (within ARM architecture), various libraries, graphics, Internet and etc.

Disadvantages: it's almost impossible to work outside Keil's environment, slightly truncated functional, only ARM platforms are supported, experience shows that it loses to many RTOS for speed.

UC/OS - a powerful commercial RTOS.

Advantages: a huge number of functions and libraries, supports multiple hardware platforms.

Disadvantages: closed commercial system, difficult to use.

QNX - POSIX-compatible real-time operating system, intended primarily for embedded systems. It is considered to be one of the best implementations of the concept of microkernel operating systems.

Advantages: it can work on almost any modern processor used in the embedded market. Among these platforms are families: x86, MIPS, PowerPC, as well as specialized processor families, such as SH-4, ARM, StrongARM and xScale, occupies a leading position among the real-time OS on the PC platform.

Disadvantages: closed commercial system, high license cost and strong dependence on QNX Software Systems in terms of licensing of developed software.

After analyzing the state of development of the market of ready real-time operating systems, it can be concluded, that for the implementation of a multi-version execution environment, FreeRTOS is preferable. Since it is ported to almost all popular platforms, it implements the basic functionality of execution environment that we need, which will greatly simplify the development of a management system based on it, and most importantly - it is distributed as an open source code available for modification (distributed under a license based on GNU GPL v2 and self-assembly and provided with documentation).

Also, FreeRTOS contains functionality for real-time work, requires minimum modifications, to implement the execution environment of multi-version software and for development of application software.

3. IDE AND FREERTOS MODIFICATIONS

From the point of view of the on-board software developer, it is advisable to have a set of unified tools that allows you to design cross-platform on-board software with obtaining the final assembly of application software on the basis of a multi-version real-time execution environment. This solution can be an integrated development environment in which the developer creates software modules that solve applied problems in both mono and multi-version options and the multi-version execution of the modules under development in real time will provide the RTOS, which is the basis of the developed environment.

Conceptually IDE is a unified means of developing components of cross-platform on-board software and its documentary support (Figure 1). The main functional modules of IDE: development of mono-version software, development of multi-version software, electronic document management, editing of the decision block, editing of FreeRTOS parameters, functional testing.

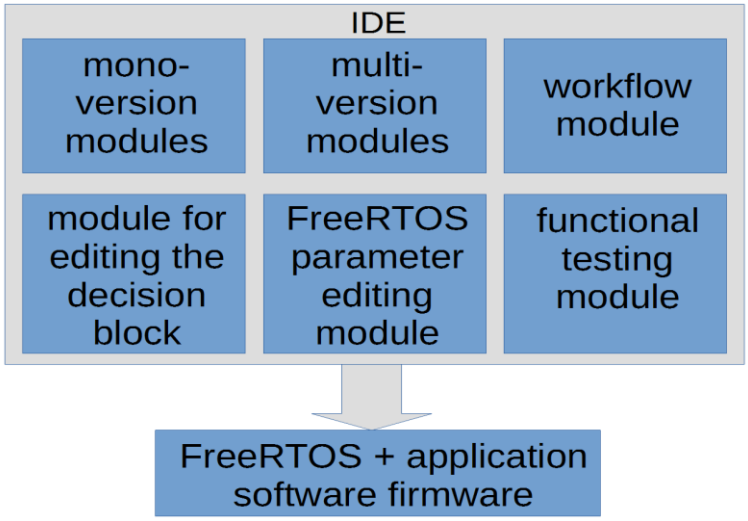

Fig. 1. Generalized structure of IDE

The output of the IDE is the firmware for the target hardware platform, which includes the RTOS and application software. At the same time, the main task is the creation of software interfaces that will ensure the implementation of the developed software modules in the RTOS in real time mode. For mono-software, the software interfaces are sufficient. For multi-version software, software interfaces are necessary for interaction not directly with the RTOS, but with a decision block, which in turn will interact with the RTOS itself via its API [7].

Let’s consider the technical features of implementing such software modules - necessary modifications to ensure the required functionality, namely - the possibility of multi-version execution of fault-tolerant software [8] in real time.

First of all, we need to create a decision block, which will also be executed as a task in the RTOS, run versions, etc.

The decision block is implemented as *static void RunComp()* function. At the beginning of this function, local variables are declared, memory allocated, if necessary, the functions of initializing arrays, etc. are called. If the data needs to be read from a file and their size is not too large to be placed in the RAM, the file is opened, the data is saved in variables and the file is closed. This is more reasonable than reading one value each time. After that, version threads are started by commands *xTaskCreate(version1, "version1", configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY + 5, &ver1);*.

The versions themselves are similar to function *static void version1()*. At the beginning of each function, local variables are declared and memory is allocated. The main difference is that the main function code is enclosed in an infinite loop *for (;;)*.The first line in the cycle is the command *vTaskSuspend(NULL);*. Thus, at the version task start, variables are declared, memory allocation and all the preparatory procedures are performed, after which the version goes into the Suspend state. From the Suspend state version will be called if necessary, prepared (ready) to calculate the task.

To transfer input data to the version and get the results, it is recommended to use the queuing mechanism, because it implements a number of functions that protect the correctness of data and increase reliability. Thus, at the beginning of the function code, the input data from the queue will be read, and at the end, the results are written to the queue. The return function cannot be used as tasks, because functions of *static void* type are used and when the function is executed correctly, the function never reaches the end [9].

Tasks are in Suspend mode, after all tasks of all versions are started. If it is necessary to execute the next iteration of this task, the input data is placed in the appropriate queue and the versions are called by the command *vTaskResume(ver1);*. After that, the versions are transferred to the execution and they continue to execute the code which follow the *vTaskSuspend(NULL);*. Then they receive the necessary data from the queue, perform the calculations, put the results in the queue, switch to the next iteration of the cycle. And again, the first command in the cycle is triggered *vTaskSuspend(NULL);* to send the version in Suspend mode.

Further, in the decision block code, you can check how many versions have already provided responses with a command *uxQueueMessagesWaiting(QueueName);*, which will return the length of the queue calculation results. Then you can read the answers of all versions with commands *xQueueReceive*, which will also clear the queue when reading it. Using the obtained results, a multi-version vote takes place, and the answer found to be correct is sent to the output.

In our case, a weighted voting algorithm by the agreed majority is used, it will allow not only to determine the correct answer from the set of version answers, but also to compare the work of the versions themselves by the sum of their weights. In the case of a strict limitation on the response time, if one of the versions does not have time to provide

a response, it can be terminated forcibly by making a vote using the replies received in time [10].

4. PRACTICAL IMPLEMENTATION OF THE REAL-TIME EXECUTION ENVIRONMENT

A multi-version real-time execution environment has been implemented to test the efficiency of the solutions proposed in the paper. The environment is implemented in the FreeRTOS v10 RTOS, the decision block, the allocated queues, the functions that implement the proposed models and algorithms that run as separate threads, are implemented in it.

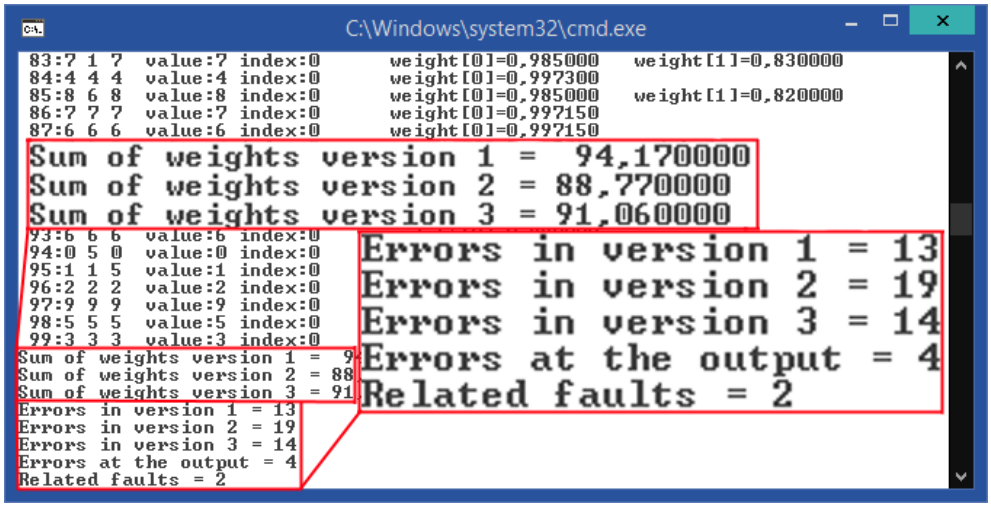

Fig. 2. The result of the voice recognition algorithms operation in the execution environment

The system was used to solve an applied problem - to increase the efficiency of the voice command recognition system for sending commands to an autonomous unmanned object. Motorized wheelchair, multi-rotor systems, and auxiliary devices for work in open space can act as such objects.

At the moment the absolutely reliable algorithms for voice recognition don't exist. All algorithms allow errors in recognition, however, different algorithms allow errors in different cases. Combining several recognition algorithms into a multi-version system can improve the reliability of voice command recognition.

The outputs of 3 ready recognition algorithms were used, on the basis of which the system chose the correct answer. The results of the program system work (Figure 2) show the effectiveness of the multi-version approach, since the quality of the system exceeds the quality of any of the algorithms used (the percentage of correctly recognized commands for the first algorithm was 87, for the second algorithm - 81, for the third

algorithm - 86, and the system correctly recognized 96% of the commands). The window displays the weights for each version, the values of which closely correspond to their reliability, which also confirms the correct operation of the system. The execution took place in real time, the results confirm the fulfillment of the system response time requirements for the event. Each iteration of multi-version voting was within 1 ms.

5. CONCLUSION

The modifications described in this paper allow us to create on the basis of FreeRTOS RTOS a multi-version real-time execution environment. That provides new tools for solving the currently critical problem of creating fault-tolerant real-time control systems. Practical implementation confirms the correctness of the proposed approach and the efficiency of all technical solutions. Verification on the real applied problem confirms the effectiveness of the multi-version approach and the correctness of the system as a whole.

REFERENCES

[1] Majid Khabbazian. Near-optimal multi-version codes. *53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2015, pp.728-732.

[2] Kulyagin V.A., Tsarev R.Yu., Prokopenko A.V., Nikiforov A.Yu., Kovalev I.V. N-version design of fault-tolerant control software for communications satellite system. *International Siberian Conference on Control and Communications (SIBCON)*, 2015, pp. 1-5.

[3] Volochiy, B., Mulyak, O., Ozirkovskyi, L., Kharchenko, V. Automation of quantitative requirements determination to software reliability of safety critical NPP I&C system. *Proceedings of the 2nd International Symposium on Stochastic Models in Reliability Engineering, Life Science, and Operations Management*, SMRLO 2016, 11 March 2016, 7433137, pp. 337-346.

[4] Kovalev, I., Losev, V., Saramud, M., Petrosyan, M. Model implementation of the simulation environment of voting algorithms, as a dynamic system for increasing the reliability of the control complex of autonomous unmanned objects, *MATEC Web of Conferences*, Volume 132, 31 October 2017, № 04011.

[5] G. Latif-Shabgahi; S. Bennett Adaptive majority voter: a novel voting algorithm for real-time fault-tolerant control systems *Proceedings 25th EUROMICRO Conference. Informatics: Theory and Practice for the New Millennium*, 1999, vol. 2, pp.113-120.

[6] Juraj Slačka, Miroslav Halás, Safety critical RTOS for space satellites, *20th International Conference on Process Control (PC)*, 2015, pp. 250 - 254

[7] Mikhail V. Saramud, Igor V. Kovalev, Vasiliy V. Losev, Peter A. Kuznetsov, Software interfaces and decision block for the execution environment of multi-version software in real-time operating systems. *International Journal on Information Technologies and Security*, No 1 (vol. 10), 2018, pp. 25-34.

[8] Tingting Hu, Ivan Cibrario Bertolott, Nicolas Navet Towards seamless integration of N-version programming in model-based design. *22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2017, pp. 1-8.

[9] Ivan Cibrario Bertolotti, RTOS support in C-language toolchains. *IEEE International Conference on Industrial Technology (ICIT)*, 2017, pp. 1328 – 1333.

[10] Juraj Slačka; Miroslav Halás Safety critical RTOS for space satellites. *20th International Conference on Process Control (PC)*, 2015, pp. 250 – 254.

Information about the authors:

Mikhail Vladimirovich Saramud - Junior Researcher of research department, Reshetnev Siberian State University of Science and Technology. Areas of Research are fault-tolerant software, system analysis;

Igor Vladimirovich Kovalev – Professor at the Department of systems analysis and operations research, Reshetnev Siberian State University of Science and Technology. Areas of Research are fault-tolerant software, system analysis;

Vasiliy Vladimirovich Losev – Associate Professor at the Department of automation of production processes, Reshetnev Siberian State University of Science and Technology. Areas of Research are automation control systems, system analysis;

Mariam Onikovna Petrosyan - Engineer of research department, Reshetnev Siberian State University of Science and Technology. Areas of Research are efficiency analysis, effectiveness analysis, decision support systems;

Manuscript received on 12 June 2018