ISSN 2413–9599 (Print)
ISSN 2541–870X (Online)

www.radioprom.org

DOI: 10.21778/2413-9599-2019-29-1-8-15

Debugging and monitoring of application programs in the BagrOS-4000 real-time operation system based on the Elbrus architecture

R.G. Gordienko[1], O.G. Fedorenko[1], A.A. Demidov[2], A.V. Fedorov[2, 3]

[1] Aviation Holding Company «Sukhoi» PJSC, Moscow, Russia
[2] MCST JSC, Moscow, Russia
[3] Institute of Electronic Control Computers named after I.S. Brook, Moscow, Russia

The article is concerned with the problems of monitoring and debugging of operating system processes, the effectiveness of which in the hard real-time operating system version does not allow any stopping to analyze the state of software and/or hardware. The paper describes the concept of a debugging and monitoring system developed taking into account this feature in the Sukhoi design bureau for the BagrOS-4000 hard real-time operating system on the Elbrus architectural platform together with the specialists of MCST JSC. The method of non-stop monitoring and data collection in hard real-time processes in the multiprocess multimodular systems is discussed. An approach to the management of debugging targets in terms of source code using the DWARF debugging information specification is presented. The transition from the instrumental machine to the system server built into the target computer is described. Given the rationale for the use of client-server architecture in the debugging and monitoring system for BagrOS-4000. A comparative analysis of the key functionality of the debugging and monitoring system with the existing debugging systems has been carried out; the key aspects of the DMS architecture have been considered. The design of a machine-dependent interface required for the integration of the independent hardware platforms into the BagrOS-4000 system when implementing the system on an integrated avionics module of the onboard complex is discussed. The results of testing of the debugging and monitoring systems are analyzed in terms of efficiency versus the classical method of using the debug console prints when debugging a real-time operating system. Most of the above solutions are universal and have been successfully tested using other microprocessor platforms on multi-threaded application programs of real-time operating systems running on multi-core processors, including the MIPS, Power PC, Intel platforms.

Keywords: real-time operating system, debugging and monitoring system, watching point, network components of the debugging and monitoring system, exception handler, machine-dependent server part, the Elbrus architecture

For citation:
Gordienko R.G., Fedorenko O.G., Demidov A.A., Fedorov A.V. Debugging and monitoring of application programs in the BagrOS-4000 real-time operation system based on the Elbrus architecture. Radiopromyshlennost, 2019, vol. 29, no. 1, pp. 8–15 (In Russian). DOI: 10.21778/2413-9599-2019-29-1-8-15

Introduction

The interaction logic of the processes of the hard real-time operating system (HRTOS) provides for monitoring and testing exclusively during the execution of these processes. Their shutdown for debugging is unacceptable, because it leads to a mismatch of the work of related processes and the inability to achieve the set debugging goals. This factor is of decisive importance in a number of critical applications, in particular in avionics, and it is presented herein as the foremost problem. Secondary problems are related to the attributes of classic debugging tools (console mode, attachment to the tool machine and the need for its exclusive use), as well as their labour intensity and limited mobility at debugging goals formation (setting the addresses of monitoring points and observed parameters).

The combination of these problems determined the formulation of the work presented in the article, which was carried out using the integrated platform including the specified operating system BagrOS-4000 and Elbrus processors.

Special requirements for debugging tools in BagrOS-4000, based on ARINC-653 and POSIX specifications, determined the need for a special approach to the organization of monitoring and debugging without complete shutdown of the application processes, but with the interruption of the operation according to BREAK instructions and on-the-fly monitoring data reading in the exception handler at a breakpoint. This approach formed the basis of a new debugging and monitoring system.

The choice in favor of the domestic manufacturer of computational tools was determined by the advanced for Russian microelectronics indicators of their modular series, which together correspond to the basic requirements of application in hard real-time. In particular, it concerns performance, power consumption, architecturally supported means of protected programming, constructive and temperature specifications.

Debugging and Monitoring Strategy

The debugging and monitoring system in HRTOS BagrOS-4000 consists of two main parts, each of which solves the foremost and secondary problems, respectively. The structure of the debugging and monitoring system is presented in Fig. 1.

The server part implements the basic strategy of the system, organizes permanent watchpoints, which are characterized by a list of monitored parameters. They are based on the BREAK instructions and give the customer the values ordered in the data list each time a point is passed. This occurs during the execution of system and application processes without their shutdown. Instead of a shutdown, an exception handler is entered (this is the mechanism chosen in the MIPS architecture [1, 2]) via BREAK, the reading of monitoring data values and exit from the handler are performed, after which the data are sent to the client by a parallel stream. At the same time, the interprocess communication can be extended in time (for the duration of the handler execution), but not stopped. This principle solves the foremost problem defined above.

DWARF (debugging with attributed record formats) is the debugging information format [3, 4] generated by the GCC compiler [5], which is used by the debugging and monitoring system to calculate automatically the addresses and types of the watchpoints, as well as the observed parameters themselves. A DWARF reader is an analyzer utility for reading the debug information and processing it for technical purposes of the system.

The network client of the debugging and monitoring system, the consumer of monitoring data, receives them from the server, establishing a TCP/IP connection with it. In fact, the client connects to the target machine, since the server of the system functions on it as part of the operating system. This client-server architecture solves the secondary problem forming a modern client implementation with a dynamic user GUI interface, including the following functionality:

- opening or closing a debugging project;
- viewing the structure of folders and project files;
- opening a project file in the source code viewer, highlighting its syntax;
- interactive (with graphic feedback) setting or removing of the watchpoint in the text of the source code;
- interactive selection in the source code window of the variable name and its inclusion in the monitoring list;
- exclusion of a variable from the monitoring list by its name;
- interactive real-time selection and monitoring of the compiled monitoring lists;
- registration in the trace files and displaying in a separate window of the history of changes of the variables obtained during their monitoring.

The problem of increasing the efficiency of client tools (secondary problem) was solved in this case with

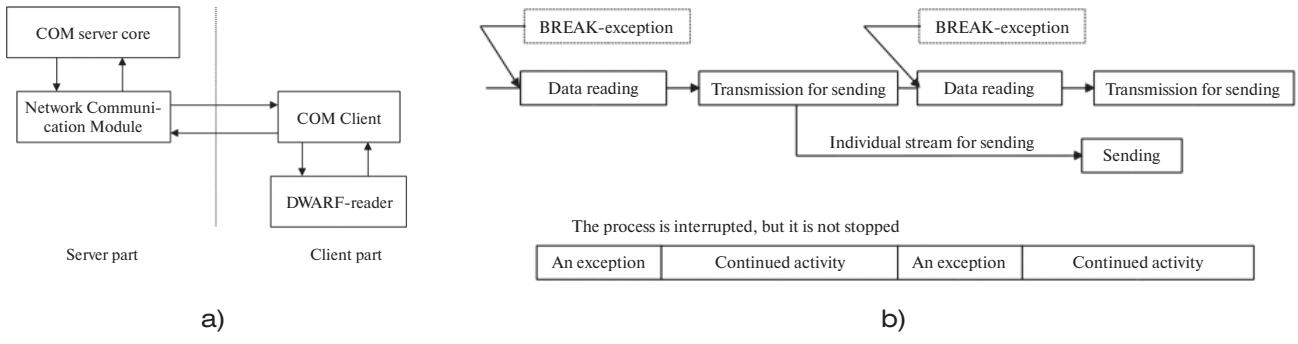

Figure 1. The principle of implementation of the debugging and monitoring system (COM) mechanisms in BagrOS-4000: a – overall structure of the system; b – parallel activity of the debugging and monitoring system mechanisms in the operating system

the help of interactive adjustment of the components of the debugging process, which are available in a modern graphical interface. The whole process of setting up the lists is carried out in terms of the source code of the software, the operation of which is to be monitored. All addresses, types of variables, and types of data locations are calculated automatically.

The main effect expected from the implementation of the debugging and monitoring system in BagrOS4000 is the ability to organize in a mobile way and monitor continuously the lists of global, local and register variables, formal parameters of the functions, and parameters of the following types:

- scalar integer and real, signed and unsigned variables;
- pointers;
- arrays;
- structures;
- associations, including composite type compositions (arrays of structures, pointers to a structure, arrays of pointers, arrays of pointers to a structure, etc.).

The system strategy is certainly not perfect: in any case, the processor of the watchpoint consumes temporary resources and shifts the pattern of temporal interactions of processes from the point when the system is working without a system. But this time is negligible compared to the usual process shutdown with classical debugging.

Nevertheless, the effectiveness of this strategy can be easily seen in the simple example of using system watchpoints instead of inserting printf into the code of debugged projects, namely:

- the use of the watchpoints does not require recompilation of the source code, as in the case of printf;
- unlike printf, the watchpoints of the debugging and monitoring system do not change the size of the image being debugged;
- the execution of printf requires considerable processor time (printf is always a resource-consuming call), while the system handler runs many times faster;
- there is no need to remove printf calls from the code upon completion of debugging.

The approaches adopted for implementation in the debugging and monitoring system are quite effective in comparison with existing solutions in terms of debugging, among which the GDB debugger that is quite close in scope can be distinguished. The analytical conclusions on this subject are presented in the table. The analysis of the system watchpoint shows that, although in some debugging tools, for example in GDB, it is possible to set the trace point, it does not meet the requirement of the foremost collection of the observed parameters at the set points without stopping the process. As for monitoring and debugging a multi-modular architecture, the following is noted: there is no debugger able to process as a single whole a field of several

Table. Comparison of selected features of real-time operating system BagrOS with the existing solutions in terms of debugging and monitoring system

The approach implemented in BagrOS-4000	Analogues	Advantages
Monitoring and debugging monitoring system points	No explicit analogues	The ability to monitor and trace in the HRTOS processes without stopping them
Client-server system architecture	Available in the GDB debugger as remote clients of the classic debugger	An open client with the ability to monitor remotely the lists of monitored parameters
Platform universality of the system	In GDB, there are versions for the Unix/Linux platforms and the WinGW version for Windows	The selected implementation in the Java Eclipse environment allows considering the Windows and Linux/Unix versions as a single project in Java source codes
The functionality of the graphic client of the system	A partial analogue is available in the GDB when using GUI emulation packages, but without hardware-dependent skins	The ability to integrate the monitoring windows and the source windows of the process being debugged. Automatic calculation from the source code of the watchpoint address and parameters. Possibility for the additional technical task to develop the interface to specialized hardware-specific skins
Monitoring and debugging of multimodular architecture	No explicit analogues	Ability in one client of the debugging and monitoring system to observe and compare the values by parameter lists from hardware modules, including by different types and models

hardware modules with the function of remote (for all modules) monitoring of the list of observed parameters.

Key aspects of the architecture and implementation of the debugging and monitoring system

The main idea of non-stop watchpoints influenced all the details of both the server and client parts. The server-part architecture is shown in Fig. 2.

Three parallel streams are involved in the implementation of the functionality: data reading and transmission in the network interaction module, as well as configuring (rebuilding) server credentials.

The data sending stream frees the exception handler from time-consuming when the monitoring data is transmitted to the client. The implementation of the dispatch process itself is noteworthy in that the server has the ability to send long data to the client as it is at the time it enters the exception handler. For this, a special copy buffer is used, where the data is placed by the handler before transmission. If the data is larger than the copy buffer size, then it will be sent without prior registration, i.e., as it is as of the time of transmission. The data buffer size is configurable; by default, it is equal to 128 KB.

The configuration stream is used to implement the addition operations (add, modify, delete) of the watchpoints and observed parameters in a pending mode. This makes it possible to manage the debugging and monitoring purposes “on-the-fly’’ and after initiation of the observation.

All monitoring processes are triggered by an exception handler at the BREAK command. It is executed in the kernel mode, so the memory for the credentials is allocated to it as a system one, so that the processor can use it regularly. The data itself is organized in such a way as to shorten the search time by the address in the list of set watchpoints and variable addresses:

- an ordered search algorithm is used to navigate in the address arrays;
- specially organized quick access tables are used to copy the observed data into the send buffer, their bypassing minimizes the time of data reading for monitoring.

These optimizations are aimed at reducing the interruption time of the monitoring process. The server architecture was also optimized in terms of demarcation of the debugging and monitoring system core links and the supported hardware.

The server has a well-defined interface for adapting the machine-dependent components. Special software adapters of the system were developed based on Elbrus-2C and Elbrus-4C processors and passed successfully a long trial operation on them, as well as on other platforms.

The key feature of the client software is its multiplatformity. The system client is developed in Java for the Eclipse platform as an additional plug-in module; the client software can run on different general-purpose operating systems without recompiling.

Such approaches demonstrate the modernity of the solution and make it possible to suggest the prospects for the development of a debugging and monitoring system in the future.

Machine-specific part

The software adapter of the debugging and monitoring system implemented on the Elbrus platform

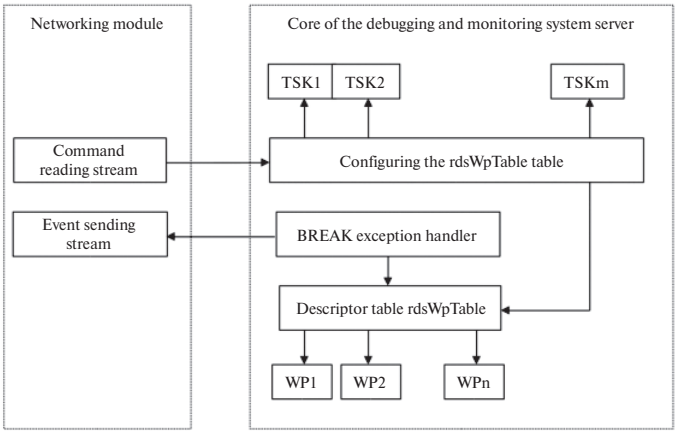

Figure 2. The server part of the debugging and monitoring system: TSK1, TSK2, TSKm – a set of task descriptors for configuring server account information (tasks for setting and removing the observation points, adding and deleting the monitored parameters, etc.); WP1, WP2, WPn – watchpoint descriptors that belong to the data structure of the rdsWpTable – table of accounts of monitoring points in the internal data of the debugging and monitoring system

corresponds to the interface of the machine-specific part of the server and supports the following functions:

- dmtaSetBreakpoint – set up a breakpoint;
- dmtaBreakpointOff – delete the breakpoint;
- dmtaRegSize – register size request;
- dmtaRegsCount – the number of registers request;
- dmtaReadCommand – read command from memory;
- dmtaIsBreakCommand – check if the command is a breakpoint;
- dmtaGetCommandSizeByVAddr – request for the size of the command at the specified address.

When adapting the machine-specific part of the system to the Elbrus platform, the following problems arose:

1. Transfer of the messages from the client to the server part from big endian to little endian.
2. Implementation of the interface functions, namely:
   - receipt of the instruction code for the virtual address;
   - registration of the setsft instructions on the virtual address;
   - handling the software trap interruption;
   - removal of the setsft instruction and recovery of the original;
   - calculation of the next executable instruction.

A TCP/IP connection is established between the client and the server, which implies the transfer of messages from big endian to little endian due to the nature of the processor architecture. This problem was solved using the builtin functions of the __builtin_bswap compiler and was built into the debugging and monitoring system.

A feature of the software implementation of the adapter system for the Elbrus platform is that the instructions of the processors of this family are packaged into broad commands that are simultaneously decrypted and executed in parallel in their own separate conveyor [6]. This is a specific feature of the architecture, which consists in the possibility to predetermine the optimal paralleling of the computational process when compiling a program.

In the adapter for the Elbrus platform, the setting and removal of the breakpoint are performed correctly based on the length of the command, replaced with the instructions set. At the level of implementation of the machine-specific server-part code, the breakpoint statement is implemented as a record of the setsft instruction to the virtual address in the USER CODE segment, which the system calculates based on the selected string in the C file and data from the DWARF table, in which for each object file contains a correspondence <source code string:: address in executable module>. The client can install setsft due to the dmtaSetBreakpoint command implemented in the server part. The setft code instruction is written by calculating the physical address from a virtual and direct copy of the instruction code to the physical address. Similarly, the removal and recovery of the original instruction (dmtaBreakpointOff and dmtaReadCommand) are performed.

During the execution of the setsft instruction, the software trap exception occurs, and the _rdsWatchpoint handler is called, which reads the observed parameters from the table and returns the original instruction code to the location, and also calculates the address of the next executable instruction, where the setsft code is written. This is necessary so that the original instruction is executed and the breakpoint itself does not disappear. After executing a Very Long Instruction Word, where the original instruction is located, the following command is executed, where the instruction with the setsft code is already installed. Next, the handler is called, from which the original instruction code will be replaced with the setsft code again, and the current executable instruction will be recovered.

The search for the next executable instruction after the breakpoint is to calculate the size of a Very Long Instruction Word. This can be done using the HS syllable header, which is the first in the command. It contains information about the length and structure of the command. The lng field indicates the number of double words in the command structure.

The debugging and monitoring system as part of the HRTOS BagrOS-4000 after the adaptation of the machine-specific part showed high efficiency on the Elbrus processors.

Prospects for the development of debugging and monitoring systems on BagrOS-4000

The main prospect of the system, which is also the main reserve of development, is the automation of monitoring, testing and tracing.

In this sense, the fundamental concept of the automation process in the system is the monitoring profile. This is a set of data stored in a file and describing a set of watchpoints, their addresses and associated groups of lists of observed parameters. The monitoring profile is saved from the IDE (integrated development environment) to a file, and then applied again when re-debugging the executable code by opening from a file. This is minimal automation.

Currently, IDE in the debugging and monitoring system is being developed as a client multiplatform software in the Eclipse environment [7, 8], developed in Java [9]. The same technologies are planned to be used in the development of IDE in the future.

The next level of development is the formation of a project to automate monitoring and testing of the P functionality in the IDE, which adds scripts to the profile, controls the setting (removal) of watchpoints, adds

(deletes) parameters in watchlists, and controls the activity of certain watchpoints depending on the value of the desired observable variables. Based on these capabilities, an effective solution is to create a library of the source code of macrotests using the built-in scripting language for manipulating the system and accessing the values of the observed variables.

The same approach will be effective to create macroscenarios to monitor the tracing in the HRTOS. Once created, the script can be saved to a file, and then opened and applied to the application code being traced. This will automatically start the code, create a trace, process and save the trace in the result file.

Conclusions

1. The authors considered a non-stop monitoring solution for multi-process multi-module systems in the HRTOS BagrOS-4000, which makes it possible to manage the debugging targets in terms of the source code, using the DWARF specification of high-level language compilers, without rebuilding the project.

2. The approach was tested on the high-performance multiprocessor platform Elbrus with practical confirmation of the effectiveness of the approach under consideration.

3. A comprehensive review of the debugging and monitoring system architecture as a software solution was carried out.

4. The interface of the system for setting up and monitoring with the machine-specific part of the debugger, the interface with the system's client and the interface with the DWARF information analysis module were analyzed.

5. The comparison of the selected characteristics of the HRTOS BagrOS with existing solutions was carried out.

6. The prospects of automation of monitoring, testing and tracing in the debugging and monitoring system were analyzed.

7. The work on the creation of IDE for HRTOS BagrOS-4000 based on the client multiplatform software in the Eclipse environment is discussed.

A certain value at the current stage of development of support for domestic avionics is the fact that the debugging and monitoring system was developed at the domestic industry enterprise for the modern developing domestic HRTOS BagrOS-4000 in integration with the domestic high-performance multiprocessor platform Elbrus.

REFERENCES

1. *IDT MIPS microprocessor family software developer's guide.* California, IDT Inc., 2005. Avaliable at: https://www.idt.com/document/mas/idt-mips-software-developers-guide-vol-1 (accessed 06.01.2019).
2. Heinrich J. *MIPS R4000 microprocessor user's manual.* 2nd ed. MIPS Technologies Inc., 1994.
3. *The DWARF debugging standard version 5.* 2017. Avaliable at: http://www.dwarfstd.org/Dwarf5Std.php (accessed 06.01.2019).
4. *Random testing for concurrency compiler bugs.* Indian Institute of Technology Kanpur, 2012.
5. Griffith A. GCC. *Polnoe rukovodstvo* [GCC. The complete reference]. Moscow, Ltd DiaSoft Publ., 2004, 624 p. (In Russian).
6. Kim A. K., Perekatov V. I., Ermakov S. G. *Mikroprotsessory i vychislitelnye kompleksy semeistva «Elbrus»* [«Elbrus» family microprocessors and computing systems]. Saint-Petersburg: Piter Publ., 2013, 272 p. (In Russian).
7. Hortsmann C., Cornell G. *Java 2. Biblioteka professionala. T. 1. Osnovy* [Java 2. Professional library. Vol. 1. The basics]. 8th ed. Moscow, Vilyams Publ., 2012, 816 p. (In Russian).
8. Silva V. *Practical eclipse rich client platform projects.* Apress, 2009. 343 p.
9. Schildt H. *Polnyj spravochnik po Java. Java SE 6 edition.* [Comprehensive guide to Java. Java SE 6 edition]. 7th ed. Moscow, Vilyams Publ., 2009, 1040 p. (In Russian).

AUTHORS

Roman G. Gordienko, head of sector, PJSC Aviation Holding Company «Sukhoi», 23B, ulitsa Polikarpova, Moscow, 125284, Russia, tel.: +7 (499) 550-01-06, e-mail: rg_gordienko@mail.ru.

Oleg G. Fedorenko, Ph.D. (Engineering), leading engineer, PJSC Aviation Holding Company «Sukhoi», 23b, ulitsa Polikarpova, Moscow, 125284, Russia, tel.: +7 (499) 550-01-06, e-mail: olegf1974@inbox.ru.

Arseniy A. Demidov, software engineer, JSC MCST, 24, ulitsa Vavilova, Moscow, 119334, Russia, tel.: +7 (499) 135-33-21, e-mail: Arseniy.A.Demidov@mcst.ru.

Aleksandr V. Fedorov, senior software engineer, JSC MCST, PJSC Brook INEUM, 24, ulitsa Vavilova, Moscow, 119334, Russia, tel.: +7 (499) 135-33-21, e-mail: Alexander.V.Fedorov@mcst.ru.

Submitted 30.10.2018; revised 20.12.2018; published online 20.02.2019.