Debugger for Real-Time OS: Challenges of Multiplatform Support

[1,3] *A.N. Emelenko <emelenko@ispras.ru>*
[1,2] *K.A. Mallachiev <mallachiev@ispras.ru>*
[1,2,3] *N.V. Pakulin <npak@ispras.ru>*
[1] *Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia*
[2] *Lomonosov Moscow State University, GSP-1, Leninskie Gory, Moscow, 119991, Russia*
[3] *Moscow Institute of Physics and Technology (State University), 9 Institutskiy per., Dolgoprudny, Moscow Region, 141701, Russia*

Abstract. In this paper, we present our work in developing a debugger for multiplatform real-time operating system Jet OS designed for civil airborne avionics. This system is being developed in the Institute for System Programming of the Russian Academy of Sciences, and it is designed to work within Integrated Modular Avionics (IMA) architecture and implement ARINC-653 API specification. Jet OS supports work on different architectures such as PowerPC, MIPS, x86 and ARM. Debugger for a real-time OS is an important tool in software development process, but debugger for RTOS is more than typical debugger used by desktop developers and we must take into account all specific features of such debugger. Moreover, we must support debugging on many platforms. However, debugger's code has to be developed for each platform and we faced the problem of porting our debugger to different architecture without developing it from scratch. In addition, the debugger must support work within emulators, because it can expand developers' capabilities and increase their efficiency. In this paper, we present the architecture of the debugger for JetOS real-time operating system, which provides capabilities for porting our debugger to a new platform in little to no time, and discuss the challenges imposed by multiplatform support in the OS.

Keywords: debugger; operating systems; multiplatform

DOI: 10.15514/ISPRAS-2017-29(4)-20

For citation: Emelenko A.N., Mallachiev K.A., Pakulin N.V. Debugger for Real-Time OS: Challenges of Multiplatform Support. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 4, 2017, pp. 295-302. DOI: 10.15514/ISPRAS-2017-29(4)-20

1. Introduction

Application debugger is an indispensable tool in developer's hands. But debugger in a real-time operating system is more than just plain debugger. In this paper we present an on-going project on debugger development for JetOS, a real-time multiplatform operating system that is being developed in the Institute for System Programming of the Russian Academy of Sciences.

JetOS is a prototype operating system for civil airborne avionics. It supports PowerPC, MIPS and x86 platforms. Also, it is designed to work within Integrated Modular Avionics (IMA) architecture and implements ARINC-653 API specification, the de-facto architecture for applied (functional) software.

The primary objectives of ARINC 653 are deterministic behavior and reliable execution of the functional software. To achieve this ARINC-653 imposes strict requirements on time and space partitioning. For instance, all memory allocations and execution schedules are pre-defined statically.

The unit of partitioning in ARINC-653 is called partition. Every partition has its own memory space and is executed in user mode. Partitions consist of one or more processes, operating concurrently, that share the same address space. Processes have data and stack areas and they resemble well-known concept of threads.

Embedded applications might be run in two different environments: in an emulator and on the target hardware. In our project, we use QEMU system emulator. Although QEMU has its own debugger support, its functionality proved to be insufficient for debugging embedded applications. Therefore, we implemented a debugger not only for the target hardware, but for the emulator as well.

2. Specific Features of Debugger for RTOS

Developing a debugger for a real-time OS is not a simple task. During developing, we faced many features of the debugger for RTOS compared to typical debuggers used by desktop developers.

Firstly, there are many interacting processes, which need to be debugged simultaneously. Therefore, our debugger must support capability to switch between them. Moreover, it needs to support work with overlapping virtual addresses space.

Secondly, it is impossible to run the debugger on the same device, where the system runs, because of the lack of on-board resources and lack of interactive facilities. That's why the debugger must be remote.

Thirdly, we must support debugging not only for application developers but also for software developers, such as drivers or kernel developers. As a consequence, the debugger can work with a highly privileged kernel and low-privilege application code.

In addition, the debugger must support work within emulators, because it can expand developers' capabilities and increase their efficiency.

Moreover, JetOS can be run on different processors, because it supports PowerPC, x86 and MIPS architecture. Consequently, the debugger has to run on these platforms

too. Thus, the debugger must consider all features of all platforms and emulators, and provide full functionality and correct execution of each process.

In order to meet this requirement, we choose GDB (GNU debugger) for the client part of the debugger, which communicates with the client part of the debugger over a serial port.

3. Related Works

We are not the first to consider the problem of debugging multiplatform RTOS. There are many types of debuggers: some of them don't have code inserted in the system, such as CodeWarrior, others use remote debugging, for example, the debugger for Pistachio microkernel; besides, there is RTOS debugger for VxWorks.

Here we briefly consider some debuggers for embedded OSes and their primary features.

3.1 CodeWarrior

CodeWarrior [3] is an IDE (integrated development environment) published by Freescale Semiconductor. It is designed to edit, compile and debug software for several microcontrollers and microprocessors (Freescale ColdFire, ColdFire+, Kinetis, Qorivva, PX, Freescale RS08, Freescale S08, and S12Z) and digital signal controllers (DSC MC56F80X and MC5680XX) used in embedded systems. It uses JTAG or BDM interface to control the target system.

CodeWarrior enables the user to debug real-time embedded applications, as well as manipulate the source code to display and change the contents of variables, arrays, and data structures. The developer can also use the debugger to work at the hardware level if necessary.

Via CodeWarrior user can:

- View and change memory, registers and variables.
- Set watchpoints.
- Set breakpoints and conditional breakpoints.
- Break on exceptions.
- Track variables

3.2 VxWorks

VxWorks [5] is a real-time operating system (RTOS) developed as proprietary software by Wind River of Alameda, California, US. It supports Intel (x86, including the new Intel Quark SoC and x86-64), MIPS, PowerPC, SH-4, and ARM architectures. Also, WxWorks includes Wind River Probe JTAG debugger, which supports the latest 32-bit and 64-bit processors based on leading architectures, such as PowerPC, ARM, Intel, and MIPS.

Wind River Probe JTAG debugger is a tool for debug application on bare metall. Developers use JTAG for target hardware communication and USB to connect to their

laptop. Probe provides capabilities to control hardware and software in a compact USB JTAG emulator.

Probe debugger implements the following set of features:

- Set hardware and software breakpoints
- Run diagnostic scripts
- Single step through code with a correlated source view
- View and modify CPU core and peripheral registers
- View and modify RAM, cache, and non-volatile memory; supports MMU

3.3 L4Ka::Pistachio

L4Ka::Pistachio [4] is the latest L4 microkernel developed by the System Architecture Group at the University of Karlsruhe. It is the first available kernel implementation of the L4 Version 4 kernel API, which provides support for both 32-bit and 64-bit architectures, multiprocessoring, and superfast local IPC. The current release supports x86-x64 (AMD64/ EM64T, K9 / P4 and higher), x86-x32 (IA32, Pentium and higher), PowerPC 32bit (IBM 440, AMCC Ebony / Blue Gene P).

Pistachio kernel uses kdbg debugger. The debugger directs its I/O via the serial line or the keyboard/screen. It is a local debugger and does not support remote debugging mode, therefore it has a very limited amount of functions.

Debugger for Pistachio can:

- Set breakpoints
- Single step
- Dump memory
- Read registers

Debugger for L4Ka::Pistachio supports two platforms, x86 and PowerPC. It is realized by dividing debugger's code into platform specific and independent parts.

Architecture dependent part of the debugger includes:

- Registers printing
- Single step support
- Memory writing
- TLB printing
- Breakpoints setting

4. Debugger's Challenges for Multiplatform Support

Our debugger consists of two parts – server and client. We use GDB for the client part of our debugger and it has the biggest part of architecture independent code.

In general, messaging mechanism between client and server doesn't change – user communicates with the client, the client sends a special-type packet to the server and waits for the server's answer. The server receives this message, checks control sum, which was sent in this packet, and if it matches the message contents, informs the

client that the message was accepted for processing. Then the server performs the action described in the packet and sends its own packet to the client. Understanding of what the client wants from the server is a part of architecture independent code in OS because almost all client requests are standardized, but their execution depends on current hardware.

We can divide our server part of the debugger into 2 parts as shown in Fig.1:

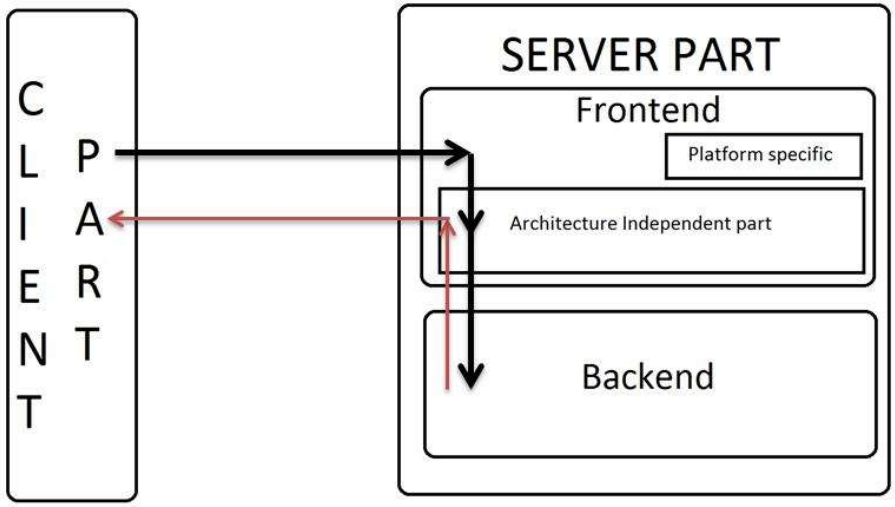

Fig. 1. Debugger's architecture.

4.1 Frontend

This part parses packets, checks control sum, calls the needed function and sends a reply.

Although almost all client-server packets are architecture independent, such requests as registers read/write depend on the target hardware. Therefore, our parser must know not only which architecture is used, but also know the type of packet the client wants to receive, for example, if the client wants to read all registers, the server must send 70 registers on PowerPC and 72 on MIPS.

4.2 Backend

This part of the debugger considers all platform capabilities and uses all available resources.

The largest part of target specific code is responsible for setting breakpoints, watchpoints, single step and read/write in memory. To implement this opportunity not only do we need special code in server part, but we must also change exception handler so that it could distinguish between debugger and regular interrupts.

For example, the single step function is realized in PowerPC architecture via special debug register, used only in this processor. Moreover, breakpoint function needs to set trap instruction where the user wants: for x86 architecture, this is 'int3' instruction, for MIPS – 'break' instruction.

The possibility of debugging applications not only on bare metal but also in emulators, such as QEMU, is also our primary goal. It adds some changes to our realization. For example, there are no debug registers in QEMU for PowerPC architecture, which is used for single step realization on bare metal. Consequently, we need special server part for each realization – on bare metal and QEMU.

5. Debugger's Capabilities

As mentioned above, all debugger's capabilities are available for all supported platforms – x86, PowerPC and MIPS.

5.1 Setting Breakpoints in Partitions and Kernel

Control execution of partitions and kernel is a key feature of debugging. It provides capabilities to more adapted debugger control mechanisms. Moreover, our system supports work with overlapping virtual address spaces, which means that debugger must correctly translate it into physical address.

5.2 Execute the Application Step-by-Step

Run application step by step is a convenient way to control system state and finding bugs. However, next instruction in code might not be the next executable extraction, for example, because of interrupt. Therefore, user can choose to stop on next instruction in code via disabling interrupts or on next executable instruction via platform capabilities.

5.3 Inspect the Application State

In each moment of time, user might want to inspect system state, i.e. memory, registers, stack trace, and threads state.

5.4 Setting Watchpoints

Watchpoints provide a great opportunity to control system state. The developer can use watchpoints to stop execution whenever the value of an expression changes, without having to predict a particular place where this may happen.

6. Debugger's Portability

As already mentioned, JetOS is being developed now, so we don't know the final count of platforms which our system will have to support.

To port our debugger on a new platform, we need only to change the process specific part of the frontend and create the backend part. All other frontend parts are the same for all processes and platforms.

It is easy to imagine the amount of work needed to port the debugger to a new platform with the help of these values:

In PowerPC server part consists of over 2000 lines of code:

- Over 1700 – frontend part
  - 1600 – architecture independent part
  - 100 – platform specific part
- Over 300 – backend part

This separation provides capabilities for porting our debugger to a new platform in little to no time.

7. Conclusion

In this paper, we presented the architecture of remote debugger for JetOS, which included architecture independent code (frontend part) and platform specific code (backend part). This architecture provides capabilities for porting debugger to a new architecture as soon as possible.

One of the next goals is to port our debugger to ARM platform, which support is being developed now.

References

[1]. Lauterbach GmbH, “RTOS debugger for VxWorks”, November 2015 http://www2.lauterbach.com/doc/rtosvxworks.pdf
[2]. Lauterbach GmbH, “RTOS-VxWorks”, 18 August 2014 http://www2.lauterbach.com/pdf/rtos_vxworks.pdf
[3]. Freescale Semiconductor, Inc. CodeWarrior Debugger, December 2, 2004 http://www.nxp.com/assets/documents/data/en/reference-manuals/Engine_PPCRM.pdf
[4]. System Architecture Group University of Karlsruhe. “The L4Ka::Pistachio Microkernel”. May 1, 2003 http://www.l4ka.org/l4ka/pistachio-whitepaper.pdf
[5]. Wind River Systems, Inc “VxWorks Product Overview”, March 2016 http://www.windriver.com/products/product-overviews/VxWorks-Product-Overview-Update.pdf
[6]. Free Software Foundation, Inc. “Debugging with gdb: the gnu Source-Level Debugger”, The Tenth Edition

Отладчик для операционной системы реального времени: проблемы мультиплатформенности

[1,3] *А.Н. Емеленко <emelenko@ispras.ru>*
[1,2] *К.А. Маллачиев <mallachiev@ispras.ru>*
[1,2,3] *Н.В. Пакулин <npak@ispras.ru>*
[1]*Институт системного программирования РАН,*
109004, Россия, г. Москва, ул. А. Солженицына, д. 25.
[2]*Московский государственный университет имени М.В. Ломоносова,*
119991, Россия, Москва, Ленинские горы, д. 1.
[3]*Московский физико-технический институт (государственный университет),*
141701, Московская область, г. Долгопрудный, Институтский переулок, д.9.

Аннотация. В этой статье мы расскажем о проекте по разработке отладчика для мультиплатформенной операционной системы реального времени JetOS, созданной для гражданских авиационных систем. Она предназначена для работы в рамках архитектуры Интегрированной Модульной Авионики (ИМА) и реализует ARINC 653 спецификацию API. Эта операционная система разрабатывается в институте системного программирования РАН, и важным шагом в ее создании является разработка инструмента для отладки пользовательских приложений. В этой статье будут рассмотрены проблемы особенностей отладчика для операционной системы реального времени и показаны методы, которыми достигается его мультиплатформенность, а также легкая переносимость на новую платформу. Более того, были рассмотрены другие отладчики для встраиваемых операционных систем, такие как CodeWarrior, отладчики для WxWorks и L4Ka::Pistachio, а также был изучен их функционал. В заключение мы представим наш отладчик, который может работать как в эмуляторе QEMU, используемом для эмуляции окружения для JetOS, так и на целевой машине на всех поддерживаемых платформах. Представленный отладчик является удаленным и построен с использованием структуры GDB, но содержит ряд расширений, специфичных для отладки встроенных приложений. Сама структура отладчика была разделена на архитектурно зависимые и независимые части, что помогает облегчить перенос отладчика на новую платформу. В то же время наш отладчик удовлетворяет большинству требований, налагаемых к отладчику операционной системы реального времени, а также уже используется разработчиками приложений для Jet OS.

Ключевые слова: отладчик; операционные системы; операционная система реального времени; мультиплатформенность.

DOI: 10.15514/ISPRAS-2017-29(4)-20

Для цитирования: Емеленко А.Н., Маллачиев К.А., Пакулин Н.В. Отладчик для операционной системы реального времени: проблемы мультиплатформенности. *Труды ИСП РАН*, том 29, вып. 4, 2017 г., стр. 295-302 (на английском языке). DOI: 10.15514/ISPRAS-2017-29(4)-20

Список литературы

[1]. Lauterbach GmbH, “RTOS debugger for VxWorks”, November 2015 http://www2.lauterbach.com/doc/rtosvxworks.pdf
[2]. Lauterbach GmbH, “RTOS-VxWorks”, 18 August 2014 http://www2.lauterbach.com/pdf/rtos_vxworks.pdf
[3]. Freescale Semiconductor, Inc. CodeWarrior Debugger, December 2, 2004 http://www.nxp.com/assets/documents/data/en/reference-manuals/Engine_PPCRM.pdf
[4]. System Architecture Group University of Karlsruhe. “The L4Ka::Pistachio Microkernel”. May 1, 2003 http://www.l4ka.org/l4ka/pistachio-whitepaper.pdf
[5]. Wind River Systems, Inc “VxWorks Product Overview”, March 2016 http://www.windriver.com/products/product-overviews/VxWorks-Product-Overview-Update.pdf
[6]. Free Software Foundation, Inc. “Debugging with gdb: the gnu Source-Level Debugger”, The Tenth Edition