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

www.radioprom.org

DOI: 10.21778/2413-9599-2019-29-1-24-30
УДК 004.451

Безопасная среда исполнения критических приложений во встраиваемых системах на базе вычислительных средств семейства «Эльбрус»

Т. Р. Мустафин[1], А. И. Алехин[1], Е. М. Кравцунов[1,2], Б. О. Макаев[1]

[1] АО «МЦСТ», Москва, Россия
[2] ПАО «Институт электронных управляющих машин им. И. С. Брука», Москва, Россия

Проблемы безопасности вычислений становятся все более актуальными в связи с объединением систем в единую глобальную сеть Интернет. Технология безопасных вычислений «Эльбрус» позволяет защитить систему от внутренних ошибок программ, а также внешних угроз. В статье приведено описание аппаратных и программных компонентов безопасной среды исполнения критических приложений во встраиваемых системах на базе вычислительных средств семейства «Эльбрус». Описана основа концепции безопасных вычислений «Эльбрус» – понятия дескриптора и внешних тегов. Рассмотрена необходимая для реализации этой среды доработка основных частей библиотеки языка Си и ядра операционной системы Linux. Исследован вопрос передачи параметров системным вызовам из приложений, выполняемых в безопасной среде. Для таких приложений проанализированы особенности процесса динамического связывания загружаемых в память модулей. При рассмотрении механизма заказов и освобождения динамической памяти представлен способ обнаружения случаев использования зависших ссылок благодаря применению дескрипторов и внешних тегов. Предложена технология контейнерной виртуализации для выявления утечек памяти в безопасной среде исполнения.

Ключевые слова: безопасный режим исполнения программ, архитектура «Эльбрус», библиотека uClibc-ng, процессорный модуль E4C/COM, ядро ОС Linux, LXC-контейнер

Для цитирования:
Безопасная среда исполнения критических приложений во встраиваемых системах на базе вычислительных средств семейства «Эльбрус» / Т. Р. Мустафин, А. И. Алехин, Е. М. Кравцунов, Б. О. Макаев // Радиопромышленность. 2019. Т. 29, № 1. С. 24–30 DOI: 10.21778/2413-9599-2019-29-1-24-30

Secure execution environment for critical applications in embedded systems based on Elbrus family computing facilities

T. R. Mustafin[1], A. I. Alekhin[1], E. M. Kravtsunov[1,2], B. O. Makaev[1]

[1] MCST JSC, Moscow, Russia
[2] Institute of Electronic Control Computers named after I. S. Brook, Moscow, Russia

The problems of security computing are becoming increasingly relevant in connection with the integration of systems into a unified global Internet. Elbrus secure computing technology allows protecting the system from the internal program errors as well as external threats. The article describes the hardware and software components of a secure execution environment for critical applications in embedded systems based on computing means of the Elbrus family. The authors describe the basis of the Elbrus secure computing concept – the idea of a descriptor and external tags. They consider the revision of the main parts of the C language library and the kernel of the Linux operating system necessary for the implementation of this environment. The study investigates the issue of passing parameters to system calls from applications running in a secure environment. For such applications, the authors consider the process of dynamic binding of the modules loaded into memory. The method for detecting cases of the use of hung links due to the use of descriptors and external tags is presented in relation to the mechanism of orders and the release of dynamic memory. The authors suggest the container virtualization technology to detect memory leaks in a safe execution environment.

Keywords: secure mode of program execution, Elbrus architecture, uClibc-ng library, E4C/COM processor module, Linux kernel, LXC container

For citation:
Mustafin T. R., Alekhin A. I., Kravtsunov E. M., Makaev B. O. Secure execution environment for critical applications in embedded systems based on Elbrus family computing facilities. Radiopromyshlennost, 2019, vol. 29, no. 1, pp. 24–30 (In Russian). DOI: 10.21778/2413-9599-2019-29-1-24-30

Введение

Применение технологии безопасных вычислений гарантирует защиту от ряда извне инициированных и неумышленных нарушений вычислительного процесса (угроз), препятствующих нормальному функционированию пользовательских и системных программ.

Представленное решение основано на аппаратной платформе «Эльбрус» и программном стеке с поддержкой безопасного режима исполнения [1]. В качестве аппаратной платформы используется процессорный модуль E4C/COM. Он построен на базе микропроцессора «Эльбрус-4С» (e2s) [2], который содержит четыре ядра архитектуры «Эльбрус», поддерживающих технологию безопасных вычислений.

Программное обеспечение среды исполнения состоит из ядра Linux с поддержкой безопасного исполнения программ и контейнерной виртуализации LXC ((LinuX Containers), безопасной реализации библиотеки языка Си (uClibc-ng), а также критических пользовательских библиотек и приложений. В статье приведены базовые средства защиты от угроз, введенные в аппаратуру платформы «Эльбрус», рассмотрены вопросы переноса основных частей библиотеки uClibc-ng в режим защищенных вычислений и обосновывается применение технологии LXC.

Базовая концепция безопасных вычислений в вычислительных средствах «Эльбрус»

Безопасность вычислений в вычислительных средствах «Эльбрус» достигается благодаря использованию дескрипторов для обращений в память. Дескриптор, включающий четыре 32-битовых слова, является обобщением понятия указателя [3]. Он описывает не только текущий адрес в памяти, но и информацию об адресуемом объекте. Структура дескриптора массива представляет собой набор полей:

- iTag (3 бита) – тип массива (внутренний тег);
- RW (2 бита) – права доступа к массиву;
- Base (59 бит) – виртуальный адрес начала массива;
- Size (32 бита) – размер массива (задается в байтах);
- Index (32 бита) – индекс адресуемого элемента (задается в байтах).

Во время обращений в память (load/store) исполняющее арифметико-логическое устройство перед

отправкой запроса в подсистему памяти проверяет права доступа RW и границы объекта Size > Index. Если в результате оказывается, что Size ≤ Index либо права доступа не соответствуют типу операции, арифметико-логическое устройство генерирует исключительную ситуацию и процесс завершается.

Защита дескриптора от изменения и формирования нового дескриптора из частей других дескрипторов достигается с помощью механизма внешних тегов. Каждому 32-битовому слову в памяти соответствует внешний двухбитовый тег. Внешний тег определяет тип слова в памяти:

- 0b00 – числовое значение;
- 0b01 – неинициализированное значение;
- 0b10 – составная часть указателя на функцию;
- 0b11 – составная часть дескриптора массива.

Внешние теги 0b10 и 0b11 генерируются аппаратурой и ядром операционной системы, пользователь может только поменять внешние теги слова на 0b00 или 0b01. При копировании части дескриптора внешний тег преобразуется в 0b00. Только при копировании целого дескриптора его теги сохраняются. Три 32-битовых слова дескриптора, содержащие поля iTag, RW, Base и Size, обладают одинаковыми внешними тегами 0b11. Четвертое 32-битовое слово дескриптора, содержащее поле Index, может свободно изменяться пользовательской программой и имеет тег 0b00. Во время обращений в память (load/store) арифметико-логического устройства перед отправкой запроса в подсистему памяти проверяет внешние теги дескриптора. Если оказалось, что внешние теги составляющих 32-битовых слов дескриптора не соответствуют 0b11, 0b11, 0b11 и 0b00, устройство генерирует исключительную ситуацию и процесс завершается.

Благодаря выполняемой арифметико-логическим устройством проверке Size > Index и защите дескриптора от изменения с помощью механизма внешних тегов эта технология позволяет защитить от угроз, построенных на ошибках переполнения буфера и выхода за границы массива.

Неинициализированные переменные обладают внешними тегами 0b01. При выполнении арифметических операций (add, mul, cmp и т.д.) арифметико-логическое устройство проверяет внешние теги входных аргументов. Если оказалось, что внешние теги не соответствуют тегам числового значения 0b00, устройство генерирует исключительную ситуацию и процесс завершается. Таким образом достигается контроль над использованием неинициализированных переменных.

Перенос библиотеки языка Си в безопасный режим исполнения

Для построения стека программного обеспечения, работающего в безопасном режиме, необходимо перевести в него стандартную библиотеку языка Си, которая обеспечивает взаимодействие между пользовательскими программами и ядром Linux.

В [4] в качестве стандартной библиотеки языка Си для переноса в безопасный режим исполнения выбрана uClibc-ng, приспособленная для использования во встраиваемых системах вследствие небольшого размера [5].

Как и другие библиотеки языка Си, uClibc-ng включает в себя как архитектурно-независимые, так и архитектурно-зависимые части. Почти все стандартные библиотечные функции архитектурно независимы. К основным архитектурно-зависимым частям относятся механизм обращения к системным вызовам, некоторые компоненты динамического загрузчика и библиотеки многопоточности, а также механизм заказов и освобождения памяти. Реализация этих составляющих архитектурно-зависимой части библиотеки uClibc-ng рассматривается в следующих разделах этой статьи.

Системные вызовы

Основная сложность реализации механизма системного вызова связана с передачей параметров. Параметры системных вызовов и возвращаемые значения передаются через регистровый файл. В безопасном режиме они делятся на два типа: целочисленные данные и указатели. Целые числа имеют размер 64 бита и занимают один двойной регистр, в то время как указатели в безопасном режиме имеют размер 128 бит и занимают один квадрорегистр (два двойных регистра). Системный вызов может иметь максимум шесть параметров.

В ядре Linux для взаимодействия с пользовательской программой через системные вызовы существует специальный вход – функция в ядре, на которую передается управление при обращении к системному вызову из пользовательской программы. Она отвечает за прием переданных через регистры параметров, в том числе указателей, и за вызов функций-обработчиков, соответствующих конкретному системному вызову. Следует отметить, что в безопасном режиме исполнения пользовательских приложений указатели являются дескрипторами, а в ядре – обычными целочисленными значениями. Данная особенность требует проводить преобразование дескрипторов в целые числа при обработке переданных параметров со стороны ядра. С этой целью в безопасном режиме были реализованы новые механизмы.

Со стороны uClibc-ng были введены специальные макросы для исполнения системных вызовов. Передача параметров организована через окно из 14 двойных регистров. Каждый параметр, будь то целое число или указатель, размещается на отдельном квадрорегистре. Такой способ передачи параметров позволяет использовать одинаковый механизм передачи параметров со стороны uClibc-ng для всех системных вызовов, а также избежать применения стека для размещения параметров. Пример передачи параметров системного вызова на регистрах показан на рис. 1.

Со стороны ядра был введен новый вход, в котором используется единый механизм обработки для большинства системных вызовов. Он основан на применении таблицы битовых масок для всех системных вызовов. Битовая маска для каждого системного вызова обозначает порядок параметров и их тип: единичный бит соответствует передаче дескриптора, нулевой бит – передаче целого числа. Расположение каждого параметра на отдельном квадрорегистре позволяет прочитать значения всех квадрорегистров и, руководствуясь битовой маской, преобразовать параметры-дескрипторы к целочисленным значениям.

Индивидуальная обработка параметров требуется только в случае системных вызовов, принимающих дескрипторы внутри структур, а также системных вызовов, возвращающих указатели. В первом случае необходимо также преобразовать эти дескрипторы к целочисленным значениям, пригодным для дальнейших операций в ядре. В случае возврата указателя из системного вызова необходимо сформировать дескриптор для использования в пользовательской программе.

Динамическое связывание загружаемых в память модулей

Для поддержки динамического связывания и работы с разделяемыми библиотеками в uClibc-ng, как и в других библиотеках языка Си, существует специальная программа, обозначаемая далее как динамический компоновщик. Он загружает необходимые для работы программы модули в память и разрешает зависимости между этими модулями.

В безопасном режиме архитектуры «Эльбрус» при запуске программы ядро сначала загружает в память только динамический компоновщик и передает ему управление. После получения управления динамический компоновщик загружает в адресное пространство процесса саму программу, все необходимые для ее работы разделяемые библиотеки и сохраняет информацию об используемых библиотеках в виде специальной структуры – цепочки загруженных модулей. Стоит отметить, что сам динамический компоновщик тоже является разделяемой библиотекой и включается в цепочку загруженных модулей, чтобы обеспечить возможность дальнейшего использования его функций другими библиотеками. После загрузки необходимых модулей динамический компоновщик производит разрешение зависимостей между модулями. Так как разные модули могут быть загружены по различным адресам в памяти, то при вызове из одного модуля функций другого модуля необходимо произвести вычисление адреса вызова. Сделать это можно только после загрузки всех модулей в память. Иначе этот процесс называется релокацией, или динамическим связыванием. При разрешении зависимостей в каждом модуле, в том числе и в самой программе, необходимо модифицировать специальную секцию исполняемого файла – глобальную таблицу смещений (got), содержащую в себе адреса функций и глобальных переменных. Процесс загрузки модулей в память, а также разрешения зависимостей в безопасном режиме имеет ряд своих особенностей, которые нужно учесть при реализации данного режима.

В первую очередь эти особенности непосредственно связаны с загрузкой модулей в память. Исполняемые файлы всегда содержат в себе как минимум два загружаемых сегмента: исполняемый и предназначенный только для чтения сегмент кода, а также не исполняемый, предназначенный для записи и чтения сегмент данных. Так как права на чтение и запись у этих сегментов отличаются, в безопасном режиме доступ к ним по одному

qr0		qr1		qr2		qr3		qr4		qr5		qr6	
dr0	dr1	dr2	dr3	dr4	dr5	dr6	dr7	dr8	dr9	dr10	dr11	dr12	dr13
sysn		arg1		arg2		arg3		arg4		arg5		arg6	

Рисунок 1. Пример передачи параметров системному вызову: qr – квадрорегистры; dr – двойной регистр; sysn – номер системного вызова; arg1, arg4, arg6 – аргументы-дескрипторы; arg2, arg3, arg5 – целочисленные аргументы
Figure 1. An example of transferring parameters to a system call: qr – quad registers; dr – double register; sysn – system call number; arg1, arg4, arg6 – descriptor arguments; arg2, arg3, arg5 – integer arguments

и тому же дескриптору невозможен. В сегменте кода содержится вся исполняемая часть файла. В сегменте данных содержатся глобальная таблица смещений (got) и глобальные переменные. Для проведения динамического связывания необходима инициализация got. Содержащиеся в got адреса функций и глобальных переменных в безопасном режиме являются дескрипторами, которые можно получить только на этапе исполнения программы. Поэтому для инициализации глобальной таблицы смещений в безопасном режиме используется специальная функция selfinit, включаемая компилятором в исполняемые файлы.

Для загрузки модулей в память в защищенном режиме динамический компоновщик применяет специальный системный вызов uselib, который размещает в отдельных областях памяти сегменты кода и данных модуля. Структура загруженного модуля в памяти показана на рис. 2.

При загрузке модуля в память динамический компоновщик возвращает необходимую информацию в виде специальной структуры:

```
typedef struct {
        void (* selfinit) ();
        char *gd;
    } mdd_t;
```

Она содержит в себе дескриптор на область данных загруженного модуля и дескриптор на функцию selfinit для инициализации глобальной таблицы смещений.

После загрузки модуля в память с помощью uselib динамический компоновщик производит вызов функции selfinit для этого модуля, чтобы инициализировать глобальную таблицу смещений. После инициализации got производится разрешение

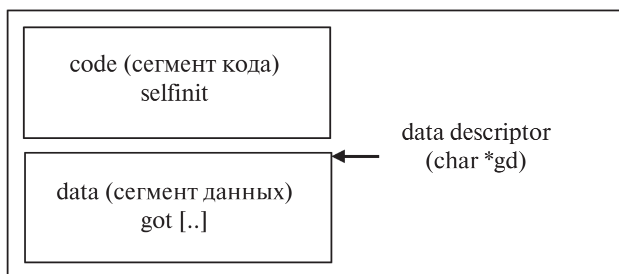

Рисунок 2. Структура загруженного в память модуля в безопасном режиме: selfinit – функция инициализации got; got [..] – глобальная таблица смещений; data descriptor (char *gd) – дескриптор сегмента данных

Figure 2. The structure of a module loaded into memory in a safe mode: selfinit – the initialization function got; got [..] – global offset table; data descriptor (char *gd) – data segment descriptor

зависимостей между модулями путем обращения к глобальной таблице смещений через дескриптор области данных.

Многопоточность

В библиотеку uClibc-ng входят две реализации библиотек многопоточности: linuxthreads и nptl. Первая по существу является устаревшей. Соответственно, при переносе библиотеки uClibc-ng в безопасный режим архитектуры «Эльбрус» в ней реализована именно nptl, включающая различные функции для работы с потоками.

В первую очередь обеспечена работа отвечающей за создание нового потока функции pthread_create, которая использует системный вызов clone. Для его работы в библиотеке uClibc-ng необходимо реализовать соответствующую функцию clone на языке ассемблера. Функция clone осуществляет обращение к системному вызову clone с передачей ему всех необходимых параметров на архитектурных регистрах, а также запуск на исполнение пользовательской функции в созданном потоке.

Обеспечена также работа функций pthread_mutex_* и pthread_join, служащих для синхронизации между потоками. Они используют универсальный системный вызов futex, а также атомарные операции. Работу атомарных операций обеспечивают специальные макросы, реализованные для безопасного режима на языке ассемблера.

Кроме того, во время создания и завершения потоков исполнения требуются функции межпроцедурного перехода setjmp и longjmp. Они реализованы на языке ассемблера специально для безопасного режима архитектуры «Эльбрус». Следует отметить, что архитектура «Эльбрус», в отличие от многих других архитектур, предусматривает межпроцедурный переход при выполнении longjmp не просто с использованием инструкции перехода, а с помощью специального системного вызова longjmp. Он реализован в ядре Linux непосредственно для архитектуры «Эльбрус».

Механизм заказов и освобождения динамической памяти

Механизм заказов и освобождения динамической памяти в языке Си дает возможность путем некорректной последовательности вызовов malloc()/free() вызвать проблемы зависших ссылок и утечки памяти. Использование зависших ссылок на ранее освобожденные массивы памяти приводит к порче новых данных. Утечки памяти могут привести к миграции памяти ответственных задач в раздел подкачки и существенному увеличению времени реакции операционной системы.

Проблема зависших ссылок в режиме безопасных вычислений решена программно-аппаратным

методом. Для этого между вызовом free() и выделением освобожденной памяти ядро операционной системы удаляет из пользовательского пространства все копии дескриптора, описывающего освобождаемую память. Благодаря наличию внешних тегов ядро отличает копию дескриптора от просто числового значения. Функция очистки из пользовательского пространства запускается системным вызовом. Переключение контекста занимает много времени, поэтому очистка запускается один раз на несколько вызовов free(). Тем не менее реальное освобождение памяти и ее выделение заново производятся только после выполнения функции очистки. Таким образом, реализована защита от зависших ссылок.

Технология контейнерной виртуализации

Проблема утечек памяти режимом безопасных вычислений не контролируется. Однако с помощью контейнерной виртуализации LXC можно обнаружить вызванные ей ошибки.

LXC представляет собой систему виртуализации на уровне операционной системы для запуска нескольких изолированных пользовательских пространств-контейнеров, которые обладают собственными пространствами процессов и сетевыми стеками, на одном компьютере [6]. При этом все контейнеры применяют один экземпляр ядра операционной системы Linux.

Эта технология, основанная на механизмах cgroups и namespaces ядра Linux [6], предоставляет возможность установить ограничения на использование ресурсов вычислительного устройства: время исполнения на CPU и потребление оперативной памяти. В то же время запуск приложения с утечками памяти внутри контейнера с ограничением оперативной памяти приведет к исчерпанию лимита памяти и аварийному завершению приложения.

Выводы

Описанная в данной работе безопасная среда исполнения критических приложений во встраиваемых системах на базе вычислительных средств «Эльбрус» предполагает реализацию обращений к памяти исключительно через дескрипторы. Развитая структура дескриптора и защита дескрипторов механизмом внешних тегов обеспечивают предохранение от угроз, основанных на ошибках переполнения буфера и выхода за границы массива. Кроме того, механизм внешних тегов позволяет отследить применение неинициализированных переменных в программах. Для программирования в безопасном режиме реализована библиотека uClibc-ng языка Си, позволяющая создавать не только синтетические тесты, но и реальные программы с использованием разделяемых библиотек и многопоточности. Программно-аппаратная реализация механизмов выделения (освобождения) динамической памяти осуществляет контроль за применением зависших указателей. Технология LXC в безопасной среде исполнения позволяет отследить утечки памяти.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Ким А.К., Перекатов В.И., Ермаков С.Г.* Микропроцессоры и вычислительные комплексы семейства «Эльбрус». СПб.: Питер, 2013. 272 с.
2. *Каталог* продукции ИНЭУМ. Официальный сайт ПАО «ИНЭУМ им. И.С. Брука» [Электронный ресурс]. URL: http://ineum.ru/files/59db40/730cd8/502916/000000/katalog_produktsii_ineum_2017ls.pdf (дата обращения: 09.01.2019).
3. *Волконский В.Ю.* Безопасная реализация языков программирования на базе аппаратной и системной поддержки // Вопросы радиоэлектроники. 2008. Т. 4, № 2. С. 98–141.
4. *Выбор* дистрибутива программного обеспечения для индустриальных и бортовых систем, использующего технологию защищенных вычислений архитектуры «Эльбрус» / Т.Р. Мустафин, А.И. Алехин, А.С. Куян, Е.М. Кравцунов, С.В. Семенихин // Вопросы радиоэлектроники. 2017. № 3. С. 44–47.
5. *Andersen E.* µClibc specifications. Официальный сайт проекта µClibc [Электронный ресурс]. URL: https://uclibc.org/specs.html (дата обращения: 09.01.2019).
6. *Graber S.* LXC 1.0: Blog post series (Stéphane Graber's website) [Электронный ресурс]. URL: https://stgraber.org/2013/12/20/lxc-1-0-blog-post-series/ (дата обращения: 09.01.2019).

REFERENCES

1. 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).
2. *Katalog produktsii INEUM. Ofitsialnyi sait PAO «INEUM im Bruka»* [Product catalog of INEUM. Website of PJSC Brook INEUM] (In Russian). Avaliable at: http://ineum.ru/files/59db40/730cd8/502916/000000/katalog_produktsii_ineum_2017ls.pdf (accessed 09.01.2019).
3. Volkonskii V. Yu. Secure implementation of programming languages and hardware based system support. *Voprosy radioelektroniki*, 2008, vol. 4, no. 2, pp. 98–141 (In Russian).
4. Mustafin T. R, Alekhin A.I., Kuyan A.S., Kravtsunov E.M., Semenikhin S.V. Software distribution choice for industry and board systems uses security computing technology of «Elbrus» architecture. *Voprosy radioelektroniki*. 2017, no. 3, p. 44–47 (In Russian).

5. Andersen E. μClibc specifications. μClibc project website. Available at: https://uclibc.org/specs.html (accessed 09.01.2019).
6. Graber S. LXC1.0: Blog post series (Stéphane Graber's website). Available at: https://stgraber.org/2013/12/20/lxc-1-0-blog-post-series/ (accessed 09.01.2019).

ИНФОРМАЦИЯ ОБ АВТОРАХ

Мустафин Тимур Рустемович, аспирант РТУ МИРЭА, инженер-программист 1 категории, АО «МЦСТ», 119334, Москва, ул. Вавилова, д. 24, тел.: +7 (499) 135-33-21, e-mail: timur.r.mustafin@mcst.ru.
Алехин Андрей Игоревич, инженер-программист, АО «МЦСТ», 119334, Москва, ул. Вавилова, д. 24, тел.: +7 (499) 135-33-21, e-mail: andrey.i.alekhin@mcst.ru.
Кравцунов Евгений Михайлович, зам. начальника отделения, АО «МЦСТ», ПАО «Институт электронных управляющих машин им. И. С. Брука», 119334, Москва, ул. Вавилова, д. 24, тел.: +7 (499) 135-33-21, e-mail: evgeniy.m.kravtsunov@mcst.ru.
Макаев Борис Олегович, инженер-программист, АО «МЦСТ», 119334, Москва, ул. Вавилова, д. 24, тел.: +7 (499) 135-33-21, e-mail: b.makaev@innopolis.ru.

AUTHORS

Timur R. Mustafin, graduate student, RTU MIREA, software engineer 1st category, MCST JSC, 24, ulitsa Vavilova, Moscow, 119334, Russia, tel.: +7 (499) 135-33-21, e-mail: timur.r.mustafin@mcst.ru.
Andrey I. Alekhin, software engineer, JSC «MCST», 24, ulitsa Vavilova, Moscow, 119334, Russia, tel.: +7 (499) 135-33-21, e-mail: andrey.i.alekhin@mcst.ru.
Evgeniy M. Kravtsunov, deputy head of department, MCST JSC, Institute of Electronic Control Computers named after I. S. Brook, 24, ulitsa Vavilova, Moscow, 119334, Russia, tel.: +7 (499) 135-33-21, e-mail: evgeniy.m.kravtsunov@mcst.ru.
Boris O. Makaev, software engineer, MCST JSC, 24, ulitsa Vavilova, Moscow, 119334, Russia, tel.: +7 (499) 135-33-21, e-mail: b.makaev@innopolis.ru.

Поступила 30.10.2018; принята к публикации 20.12.2018; опубликована онлайн.
Submitted 30.10.2018; revised 20.12.2018; published online.