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

www.radioprom.org

DOI: 10.21778/2413-9599-2020-30-1-47-53
УДК: 004.4'2

Поддержка работы со звуковой подсистемой ALSA в бинарном компиляторе уровня приложений x86→«Эльбрус»

Е. С. Носкова[1, 2], А. Ф. Рожин[2, 3]

[1] Московский физико-технический институт (национальный исследовательский университет), Москва, Россия
[2] АО «МЦСТ», Москва, Россия
[3] ПАО «Институт электронных управляющих машин им. И. С. Брука», Москва, Россия

Успешность аппаратной платформы во многом определяется набором доступного на ней программного обеспечения. Именно поэтому очень актуальной является задача расширения базы приложений, запускаемых на платформе «Эльбрус». Один из способов ее решения – использование технологии бинарной компиляции. Работа многих современных приложений так или иначе связана с обработкой звука. Одной из наиболее популярных библиотек для работы со звуковыми устройствами в операционных системах семейства Linux на сегодняшний день является библиотека ALSA. Ее функциональность базируется на наборе команд системного вызова ioctl, которые в бинарном компиляторе x86→«Эльбрус» до недавнего времени не поддерживались. Поэтому возникла задача реализации поддержки команд ALSA-ioctl в этом бинарном компиляторе. Для ее решения были изучены средства, при помощи которых эта технология реализована в ядре Linux. Полученные знания были применены для разработки собственных для бинарного компилятора средств, которые сделали возможным использование ALSA в транслируемых под архитектуру «Эльбрус» приложениях.

Ключевые слова: бинарная трансляция, ALSA, бинарный компилятор уровня приложений, Linux, архитектура x86, архитектура Эльбрус

Для цитирования:
Носкова Е. С., Рожин А. Ф. Поддержка работы со звуковой подсистемой ALSA в бинарном компиляторе уровня приложений x86→«Эльбрус» // Радиопромышленность. 2020. Т. 30, № 1. С. 47–53. DOI: 10.21778/2413-9599-2020-30-1-47-53

Support for working with the ALSA sound subsystem in a binary compiler of application level x86→«Elbrus»

E.S. Noskova[1, 2], A.F. Rozhin[2, 3]

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

The success of a hardware platform is mainly determined by the software available for it. That is why the problem of enlarging the number of applications launched on the «Elbrus» platform is very acute. One of its possible solutions is the usage of binary compilation technology. The work of many modern applications is somehow related to sound processing. One of the most popular modern libraries for working with sound devices in Linux operating systems is the ALSA library. Its functionality is based on the set of ioctl system call commands. They were not supported in the x86→«Elbrus» binary compiler until recently. Therefore, occurred the issue of implementing support for ALSA-ioctl commands in binary compiler. In order to solve this problem, it was necessary to study the Linux kernel implementation of this technology. The gained knowledge was used to develop proprietary tools for the binary compiler making it possible to use the ALSA in applications translated under the «Elbrus» architecture.

Keywords: binary translation, ALSA, application level binary compiler, Linux, x86 architecture, «Elbrus» architecture

For citation:
Noskova E.S., Rozhin A.F. Support for working with the ALSA sound subsystem in a binary compiler of application level x86→«Elbrus». Radio industry (Russia), 2020, vol. 30, no. 1, pp. 47–53. (In Russian). DOI: 10.21778/2413-9599-2020-30-1-47-53

Введение

Задачу запуска приложения, осуществляющего работу со звуком, на аппаратной платформе «Эльбрус» можно решить с помощью портирования приложения, скомпилированного при помощи языкового компилятора под архитектуру «Эльбрус». Данный подход реализуем, если у приложения имеются исходные коды. Однако как в процессе сборки, так и при запуске приложения могут возникнуть разного рода ошибки. Например, в коде приложения может присутствовать архитектурно-зависимый код или используемые библиотеки могут быть еще не перенесены на платформу «Эльбрус»; также возможны ошибки, вызванные неправильной работой языкового компилятора или приложения. Устранение подобных проблем может занять неопределенное время, но если времени нет, можно рассмотреть такой вариант, как запуск при помощи бинарного компилятора. Этот же вариант подойдет и при отсутствии исходных кодов приложения или используемых им библиотек.

Таким образом, становится актуальной возможность запуска приложения, осуществляющего работу со звуком, на платформе «Эльбрус» при помощи бинарного компилятора приложений. Частным случаем запускаемого приложения может быть приложение, использующее библиотеку ALSA (Advanced Linux Sound Architecture) для обработки звука. Для корректной работы подобных приложений под бинарным компилятором необходимо, чтобы компилятор мог обрабатывать запросы, приходящие от ALSA. Поэтому наличие в компиляторе приложений поддержки ALSA очень важно для запуска многих приложений. Данная статья посвящена реализации поддержки работы с современной звуковой архитектурой ALSA, которая на сегодняшний день является наиболее распространенной, в бинарном компиляторе уровня приложений x86→«Эльбрус».

Интерфейс ALSA

ALSA – архитектура звуковых драйверов, обеспечивающая функциональность аудио и MIDI для операционной системы (ОС) Linux [1]. Главными особенностями ALSA являются:

- эффективная поддержка всех типов аудиоинтерфейсов: от потребительских звуковых карт до профессиональных многоканальных аудиоинтерфейсов;
- полностью модульные звуковые драйверы;
- поддержка драйвера Open Sound Subsystem (OSS), обеспечивающего бинарную совместимость с большинством программ, использующих OSS.

Для упрощения программирования приложений с использованием ALSA и обеспечения более высокого уровня функциональности был реализован специальный интерфейс – библиотека пользовательского пространства (libasound). Библиотека libasound содержит набор функций, с помощью которых приложение может обращаться к звуковым устройствам для выполнения каких-либо действий [2]. На данный момент в библиотеке реализованы следующие интерфейсы:

- Information Interface,
- Control Interface,
- Mixer Interface,
- PCM Interface,
- Raw MIDI Interface,
- Sequencer Interface,
- Timer Interface.

Бинарный компилятор

Бинарная, или двоичная, компиляция – высокопроизводительное и надежное средство обеспечения переносимости двоичных кодов между вычислительными машинами, реализованными на базе различных процессорных архитектур [3]. Для архитектуры «Эльбрус» предназначены два бинарных компилятора, используемые для разных целей: трансляторы уровня приложений и уровня виртуальной машины.

Для ясности дальнейшего изложения определим понятие исходной платформы. Исходной назовем такую аппаратную платформу, приложения которой предполагается запускать при помощи бинарного компилятора на платформе с другой процессорной архитектурой. В данной статье в качестве исходной рассматривается платформа x86 и ее расширение x86_64.

Наиболее полно задача двоичной совместимости решается универсальным транслятором уровня виртуальной машины, обеспечивающим возможность запуска любых ОС и приложений исходной платформы. В то же время для обеспечения возможности одновременного запуска как нативных, так и x86-приложений на вычислительных комплексах, созданных на базе микропроцессоров семейства «Эльбрус», подходит компилятор уровня приложений. Бинарный компилятор уровня приложений x86→«Эльбрус» является приложением ОС «Эльбрус» (Linux, портированный под «Эльбрус») и позволяет запускать пользовательские приложения в x86-кодах, функционирующие на исходной платформе под управлением ОС семейства Linux.

Бинарный компилятор, исполняя x86-приложение, динамически преобразует x86-код в эквивалентный ему код архитектуры «Эльбрус», а затем выполняет его. На данный момент в рамках двоичного транслятора реализовано четыре возможных метода исполнения кодов исходной платформы, предназначенных для разных целей [4, 5].

При исполнении x86-кода обращения в память, исходящие из логики работы x86-приложения, не должны нарушать целостность данных бинарного компилятора. Для изоляции пространства адресов, используемых x86-приложением при его запуске через бинарный компилятор, от адресов, используемых бинарным компилятором, в архитектуре «Эльбрус» введено два виртуальных пространства: первичное – виртуальное пространство нативных приложений – и специальное вторичное, используемое бинарными компиляторами. Для компилятора приложений вторичное пространство выделяется в виртуальном пространстве памяти компилятора приложений, которое рассматривается как первичное пространство, и располагается в нем с некоторым смещением (более 4 Гб). Величина смещения зависит от поколения процессора «Эльбрус». Учитывая то, что компилятор приложений является 32-битным, наличие смещения гарантирует непересечение вторичного пространства и диапазона адресов, относящихся непосредственно к бинарному транслятору. Существование вторичного пространства делает невозможным изменение x86-приложением данных компилятора приложений, но оставляет возможность обращаться к памяти x86-приложения как бинарному компилятору, так и ядру ОС.

На рис. 1 изображена схема виртуальной памяти запущенного компилятора приложений. Показано взаимное расположение вторичного пространства и памяти компилятора приложений. Под собственной

Рисунок 1. Схема виртуальной памяти компилятора приложений
Figure 1. Application compiler virtual memory diagram

памятью компилятора приложений понимается память, доступная ему через 32-битный указатель.

Работа с устройствами в ОС семейства Linux

В ОС семейства Linux устройства представляются в виде файлов специальной файловой системы /dev. Работа с ними осуществляется при помощи системного вызова ioctl. Он принимает на вход: дескриптор файла (который является результатом системного вызова open), представляющего устройство в файловой системе; код команды, которую необходимо выполнить; необязательный параметр – аргумент (структура, число) команды [6]. Числовое значение команды формируется из нескольких составляющих: кода символа, обозначающего принадлежность к определенному классу устройств; размера необязательного аргумента (0 – если его нет); типа команды, определяющего направление передачи данных (в ядро – чтение, из ядра – запись, в обе стороны – чтение и запись); порядкового номера команды, уникальной для данного класса устройств [7].

Все вышесказанное применимо, в частности, и к звуковым устройствам. На рис. 2 показаны составляющие кода команды ALSA-ioctl на примере SNDRV_RAWMIDI_IOCTL_PARAMS_GUEST. Код команды составляется в результате работы макроса, который принимает на вход все заявленные составляющие.

Проблемы поддержки ALSA в бинарном компиляторе

В процессе решения задачи возник ряд сложностей, обусловленных особенностями устройства бинарного компилятора.

Проблема 1. Возможное совпадение числового значения ioctl-команд для разного типа устройств

К сожалению, в ядре Linux код символа для обозначения различных типов устройств не является уникальным. Существуют устройства, например устройства типа USB и звуковые Control-устройства, которые имеют одинаковые идентификаторы, что может приводить к коллизии по коду ioctl-команд. Из-за этого при обработке ioctl-запроса от x86-приложений необходимо учитывать не только ioctl-команду, но и особенности файлового дескриптора (тип этого устройства), также являющегося аргументом системного вызова. Данная проблема является специфической для бинарного компилятора, так как в нем, в отличие от ядра Linux, определение типа ioctl-команды производится по буквенному обозначению, а не по файловому дескриптору.

Проблема 2. Необходимость преобразования адресов, приходящих из x86-приложения

В большинстве случаев при исполнении системного вызова ioctl на вход в ядро передается так называемый аргумент этого системного вызова – указатель на специфическую структуру.

Из устройства адресного пространства компилятора приложений, которое было описано выше, следует, что указатели, передаваемые через аргумент в ядро, нуждаются в преобразовании в полный линейный 64-битный адрес из адреса вторичного пространства. Также в случае, когда аргумент ioctl содержит указатель на массив указателей, простое преобразование адреса вторичного пространства (по которому осуществляется доступ к массиву указателей) к полному линейному адресу будет недостаточным. Кроме этого, необходимо преобразовать все указатели, входящие в массив указателей. Оба этапа преобразования играют существенную роль, так как размеры указателей для 64 и 32 бит различны и, следовательно, при игнорировании любого из этапов адрес будет указывать на неверное место в памяти.

Проблема 3. Ограничение применения специально созданных для 32-битных приложений compat ioctl из-за несоответствия размеров указателей

64-битные версии ОС семейства Linux, как правило, позволяют запускать не только 64-битные

Тип команды / Command type	Размер аргумента-структуры / Argument size	Код символа / Character code	Порядковый номер команды / Command sequence number
WR	sizeof (apl_RawmidiParamsStructGuest_t)	'W'	0x10
31	29	15	7

Рисунок 2. Структура команды ALSA-ioctl на примере SNDRV_RAWMIDI_IOCTL_PARAMS_GUEST со значением 0xc0305710
Figure 2. ALSA-ioctl command structure using SNDRV_RAWMIDI_IOCTL_PARAMS_GUEST with the value 0xc0305710 as an example

пользовательские приложения, но и 32-битные. Однако запуск последних требует специфической обработки элементов аргумента-структуры. Например, структура, содержащая поле типа long, будет иметь разные размеры с точки зрения x86-приложения и с точки зрения 64-битного ядра ОС «Эльбрус» (например, sizeof_x86(long) = 4, sizeof_elbrus(long) = 8) [8]. Данная проблема обычно решается использованием compat-системных вызовов. Они существуют для запуска 32-битных приложений со своими внутренними преобразованиями аргумента. Однако компилятор приложений не всегда может использовать compat-системные вызовы, реализованные в ядре Linux. Как было отмечено в описании проблемы 2, все указатели, передаваемые в ядро, преобразуются в 64-битные, а compat-системные вызовы могут принимать на вход только 32-битные адреса.

Таким образом, возникает необходимость производить преобразования над аргументами в специально реализованных функциях во избежание несовпадения кода ioctl-команд для x86-приложения и ядра.

Проблема 4. Контроль отсутствия защиты на запись для участков памяти, потенциально подверженных изменению со стороны ядра

Обработка аргумента системного вызова ioctl может включать в себя еще одну важную процедуру – снятие защиты на запись для некоторых участков памяти, на которую ссылается данный аргумент. Эта особенность связана исключительно с бинарным компилятором x86→«Эльбрус».

Код архитектуры «Эльбрус», полученный при трансляции x86-кода, должен обладать обратной совместимостью со своим x86-оригиналом. Это значит, что если по какой-то причине изменился x86-код, для которого ранее был создан эквивалентный код архитектуры «Эльбрус», то бинарный компилятор более не вправе использовать последний, так как в общем случае он уже не соответствует текущему x86-коду [9]. Поэтому при трансляции на x86-код накладывается «защита». Этот механизм осуществляется за счет того, что в операционных системах (для более эффективного использования физической памяти) память запускаемого приложения разбивается на набор регионов постоянного размера, так называемых страниц, которые описываются специальными атрибутами [10]. Идея заключается в том, что страницы памяти, накрывающие x86-код, для которого создана трансляция, помечаются как недоступные для записи. Если приложение все же захочет изменить свой код или данные, расположенные на одной странице с «защищенным» кодом, то компилятор приложений получит от ядра ОС сигнал об исключительной ситуации – попытке записи в страницу, для которой это не разрешено. В этом случае бинарный компилятор удалит все ранее созданные трансляции, чьи x86-оригиналы хотя бы частично пересекаются с «защищенной» страницей, затем снимет с нее «защиту» и произведет прерванную запись. Благодаря такому механизму обеспечивается исполнение исключительно корректного кода. Однако все это работает, только когда на процессоре исполняются коды компилятора приложений. При попытке записи в «защищенную» страницу со стороны ядра, например при исполнении того или иного системного вызова, произойдет аварийное завершение работы компилятора приложений.

Практическое решение возникших проблем

Проблема 1

С учетом описанных трудностей авторами было предложено следующее решение для поддержки ALSA:

- в бинарном компиляторе был создан отдельный ioctl-обработчик для запросов от ALSA, представляющий из себя набор собственных функций (для каждого типа устройств) в виде конструкции switch по поддерживаемым ioctl-командам, в которых происходило необходимое преобразование аргумента ioctl для последующей передачи его в ядро ОС «Эльбрус»;
- переход в созданный обработчик осуществлялся только для файлов из директории /dev/snd, так как в ней находятся все файлы звуковых устройств;
- при попытке исполнить незвуковой ioctl (запрос к файлу из /dev/snd) его обработка переходила в общий switch для ioctl-запросов.

Проблемы 2 и 3

Для запуска 32-битного приложения было создано два экземпляра одной структуры – 32-битная для работы с приложением и 64-битная для передачи в ядро, – а также функции конвертации, в которых поля одной структуры присваивались полям другой с соответствующими преобразованиями, если необходимо. Преобразования делились на два типа: преобразования указателей к вторичному пространству и обратные, которые также были необходимы и для запуска 64-битных приложений, и для преобразования типов.

Для структур с двойными указателями было создано два массива: один – под 64-битные, второй – под 32-битные указатели в памяти компилятора приложений. Во второй массив в дальнейшем копировались 32-битные указатели из памяти x86-приложения (рис. 3), затем для каждого указателя из массива производилась отдельная конвертация.

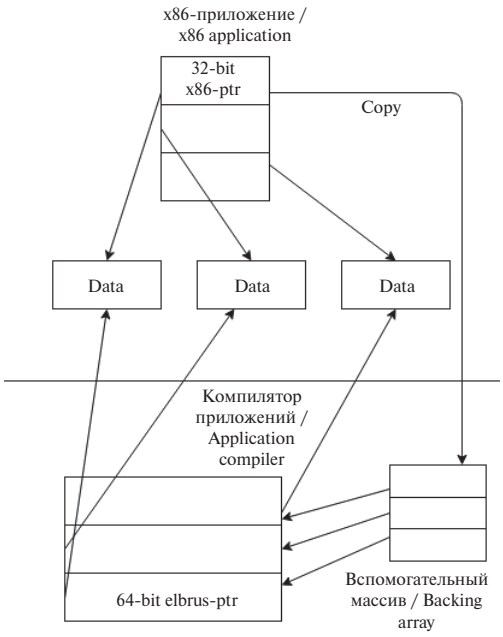

Рисунок 3. Конвертация аргументов с двойными указателями
Figure 3. Converting double-pointer arguments

Таким образом, аргумент, получаемый системным вызовом из x86-приложения, сначала преобразуется в 64-битный формат для ОС «Эльбрус», а затем отправляется в ядро. В результате системного вызова в аргументе (структуре) ядро может вернуть определенные данные в формате для ОС «Эльбрус». Чтобы эти данные были получены и корректно обработаны x86-приложением, необходимо выполнить обратное преобразование из формата для ОС «Эльбрус» – в формат x86.

В качестве решения проблемы compat-системных вызовов реализованы compat-функции конвертации, в теле которых производится пересчет необходимых полей. Эти функции должны исполняться только в том случае, когда компилятору на вход поступает 32-битное приложение.

Проблема 4

Во избежание выхода из строя компилятора приложений при попытке записи в «защищенную» страницу, в теле функций конвертации вызывался ранее разработанный механизм, разрешающий эту ситуацию. Он позволяет снимать «защиту» и запрещать ее установку для некоторых участков памяти, в которых расположены данные, потенциально подверженные изменениям со стороны ядра, на время исполнения системного вызова. Поэтому снятие защиты также является одной из составляющих преобразования данных для спуска их в ядро Linux.

Рассмотрим процесс обработки аргумента ALSA-ioctl на примере команды SNDRV_PCM_IOCTL_READI_FRAMES_GUEST. При выполнении команды данные из пользовательского пространства передаются в ядро Linux, то есть происходит запись звука. Аргументом данной команды является структура, состоящая из трех полей:

1. указатель на буфер, в который будет производиться запись, – поле для записи ядром Linux;
2. максимально возможное количество считанных фреймов (фрагментов звукового потока) – поле для чтения ядром Linux;
3. количество записанных в результате фреймов – поле для записи ядром Linux.

Из состава аргумента следует необходимость создания функции конвертации, преобразующей указатель, пришедший из приложения, к указателю вторичного пространства для передачи его в ядро. При этом необходимо также снимать «защиту» на запись, так как, исходя из предназначения команды, ядро будет заполнять структуру, поданную через указатель.

Выводы

Было проведено исследование работы со звуковыми устройствами в ядре ОС Linux и изучены средства, с помощью которых в нем была поддержана работа со звуковой архитектурой ALSA. Полученные сведения были применены при разработке собственных для компилятора приложений средств, которые сделали возможным использование ALSA в транслируемых под архитектуру «Эльбрус» приложениях. Иными словами, была реализована поддержка ALSA в компиляторе уровня приложений x86→«Эльбрус». Полученное решение было протестировано на различном программном обеспечении, использующем библиотеку ALSA. Все тесты были пройдены успешно.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Alsa* project. [Электронный ресурс]. URL: https://alsa-project.org (дата обращения: 23.01.2020).
2. *Lin J.M., Cheng W.G., Fang G.M.* Software integration for applications with audio stream. 2008 International Conference on Intelligent Information Hiding and Multimedia Signal Processing. IEEE, 2008, pp. 1126–1129.
3. *Система* динамической двоичной трансляции x86→«Эльбрус» / Н.В. Воронов, В.Д. Гимпельсон, М.В. Маслов, А.А. Рыбаков, Н.С. Сюсюкалов // Вопросы радиоэлектроники. 2012. Т. 4. № 3. С. 89–108.
4. *Kanhere A.S.* Instruction and logic to perform dynamic binary translation: патент 9417855. США. 2016.
5. *Das A.* Support for a non-native application: патент 9766911. США. 2017.

6. *Corbet J., Rubini A., Kroah-Hartman G.* Linux Device Drivers, 3rd Edition. USA, O'Reilly Media, Inc., 2005, 640 p.
7. *Чезати М., Бовет Д.* Ядро Linux. СПб.: БХВ-Петербург, 2007. 1105 с.
8. *Intel→* 64 and IA-32 Architectures Software Developer's Manual, vol. 3, Intel Corp., 2006. [Электронный ресурс]. URL: https://www.intel.ru/content/www/ru/ru/architecture-and-technology/64-ia-32-architectures-software-developer-system-programming-manual-325384.html (дата обращения: 22.10.2019).
9. *Akshintala A., Jain B., Tsai Ch., Ferdman M., Porter D.* x86–64 instruction usage among C/C++ applications. In: Proceedings of the 12th ACM International Conference on Systems and Storage (SYSTOR '19). New York, Association for Computing Machinery, 2019, pp. 68–79.
10. *Остин Т., Таненбаум Э.* Архитектура компьютера. СПб.: Питер, 2013. 816 с.

REFERENCES

1. Alsa project. Available at: https://alsa-project.org (accessed 23.01.2020).
2. Lin J.M., Cheng W.G., Fang G.M. Software integration for applications with audio stream. *2008 International Conference on Intelligent Information Hiding and Multimedia Signal Processing. IEEE,* 2008, pp. 1126–1129.
3. Voronov N.V., Gimpelson V.D., Maslov M.V., Rybakov A.A. Syusyukalov N.S. Dynamic binary translation system x86→ «Elbrus». *Voprosy radioelektroniki,* 2012, vol. 4, no. 3, pp. 89–108. (In Russian).
4. Kanhere A.S. Instruction and logic to perform dynamic binary translation: patent 9417855, USA, 2016.
5. Das A. Support for a non-native application: patent 9766911, USA, 2017.
6. Corbet J., Rubini A., Kroah-Hartman G. *Linux Device Drivers, 3rd Edition.* USA, O'Reilly Media, Inc., 2005, 640 p.
7. Chezati M., Bovet D. *Yadro Linux* [Linux kernel]. Saint Petersburg, BKHV-Peterburg Publ., 2007, 1105 p. (In Russian).
8. Intel→ 64 and IA-32 Architectures Software Developer's Manual, vol. 3, Intel Corp., 2006. Available at: https://www.intel.ru/content/www/ru/ru/architecture-and-technology/64-ia-32-architectures-software-developer-system-programming-manual-325384.html (accessed 22.10.2019).
9. Akshintala A., Jain B., Tsai Ch., Ferdman M., Porter D. x86–64 instruction usage among C/C++ applications. *In: Proceedings of the 12th ACM International Conference on Systems and Storage (SYSTOR '19).* New York, Association for Computing Machinery, 2019, pp. 68–79.
10. Ostin T., Tanenbaum Eh. *Arkhitektura komp'yutera* [Computer architecture]. Saint Petersburg, 2013, 816 p. (In Russian).

ИНФОРМАЦИЯ ОБ АВТОРАХ

Носкова Елизавета Сергеевна, студент, Московский физико-технический институт (национальный исследовательский университет), 141701, Московская обл., г. Долгопрудный, Институтский пер., д.9., тел.: +7 (916) 797-90-23, e-mail: noskova_e@mcst.ru.
Рожин Александр Феодосьевич, к.ф.-м.н., начальник отдела, АО «МЦСТ», ПАО «ИНЭУМ им. И.С. Брука», 119334, Москва, ул. Вавилова, д.24, тел.: +7 (499) 135-14-75, e-mail: rozhinaf@mcst.ru

AUTHORS

Elizaveta S. Noskova, student, Moscow Institute of Physics and Technology, 9, Institutskii pereulok, g. Dolgoprudnyi, Moskovskaya obl., 141701, Russia, tel.: +7 (916) 797-90-23, e-mail: noskova_e@mcst.ru.
Aleksandr F. Rozhin, Ph.D. (Physics and Mathematics), head of department, JSC MCST, Scientific secretary of Institute of Electronic Control Computers named after I.S. Brook, 24, ulitsa Vavilova, Moscow, 119334, tel.: +7 (499) 135-14-75, e-mail: rozhinaf@mcst.ru

Поступила 30.10.2019; принята к публикации 10.11.2019; опубликована онлайн 25.02.2020.
Submitted 30.10.2019; revised 10.11.2019; published online 25.02.2020.