DOI: 10.15514/ISPRAS-2019-1(3)-23

Семейство операционных систем КЛОС

[1]*И. Б. Бурдонов, ORCID: 0000-0001-9539-7853, igor@ispras.ru*
[1]*А.С. Косачев, ORCID: 0000-0001-5316-3813 <kos@ispras.ru>*
[1,2,3]*А. К. Петренко, ORCID: 0000-0001-7411-3831, <petrenko@ispras.ru>*
[1,2,3,4]*А. В. Хорошилов, ORCID: 0000-0002-6512-4632, <khoroshilov@ispras.ru>*
[1,2]*В. Ю. Чепцов, ORCID: 0000-0003-3931-101X, <cheptsov@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]*НИУ Высшая школа экономики, 101978, Россия, г. Москва, ул. Мясницкая, д. 20*
[4] *Московский физико-технический институт, Россия, 141701, Московская область, г. Долгопрудный, Институтский пер., 9*

Аннотация. КЛОС — это технология и семейство встраиваемых операционных систем с поддержкой многоядерности, с повышенными требованиями к безопасности (КТ-178C) и защищённости (РБПО). В КЛОС обеспечивается пространственная (по памяти) и временна́я (по гарантиям времени отклика) изоляция функционального и системного программного обеспечения. Накладные расходы со стороны ОСРВ минимизированы за счёт статического конфигурирования памяти и непериодических таймеров с квантованием. В статье кратко описывается история работ по созданию операционных систем на основе микроядерного подхода, которые были начаты еще под руководством академика В.П. Иванникова в 70-е годы 20-го века и развитие, которое они получили в настоящее время. Более подробно описываются основные архитектурные решения, использованные в версиях КЛОС, которые разрабатываются в ИСП РАН для систем аэрокосмической техники в последние десять лет.

Ключевые слова: микроядерные операционные системы, изоляция приложений, детерминированное выполнение приложений, встроенные системы, системы реального времени.

Для цитирования: Бурдонов И. Б., Петренко А. К., Хорошилов А. В., Чепцов В. Ю. Семейство операционных систем КЛОС. Труды ИСП РАН, том 37, вып. 3, 2025 г., стр. 325-354. DOI: 10.15514/ISPRAS–2023–37(3)-23

CLOS Operating System Family

[1]*I. B. Burdonov, ORCID: 0000-0001-9539-7853, igor@ispras.ru*
[1]*A.S. Kossatchev, ORCID: 0000-0001-5316-3813 <kos@ispras.ru>*
[1,2,3]*A. K. Petrenko, ORCID: 0000-0001-7411-3831, <petrenko@ispras.ru>*
[1,2,3,4]*A. V. Khoroshilov, ORCID: 0000-0002-6512-4632, <khoroshilov@ispras.ru>*
[1,2]*V. Yu. Cheptsov, ORCID: 0000-0003-3931-101X, <cheptsov@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]*National Research University, Higher School of Economics 20, Myasnitskaya ulitsa, Moscow, 101978, Russia*
[4] *Moscow Institute of Physics and Technology, 9, Institutskiy per., Dolgoprudny, Moscow Region, 141701, Russia.*

Abstract. CLOS is a technology and family of embedded operating systems supporting multi-core architectures, with enhanced safety (DO-178C) and secure software development lifecycle (SSDL) requirements. KLOS provides spatial (memory) and temporal (response time guarantees) isolation of functional and system software. RTOS overhead is minimized through static memory configuration and non-periodic timers with quantization. This article briefly describes the history of work on operating systems based on the microkernel approach, which began under the leadership of Academician V.P. Ivannikov in the 1970s and the development they have undergone to date. The main architectural solutions used in KLOS versions developed at the Institute of System Programming of the Russian Academy of Sciences for aerospace systems over the past ten years are described in more detail.

Keywords: microkernel operating systems, application isolation, deterministic application execution, embedded systems, real-time systems/

For citation: Burdonov I. B., Petrenko A. K., Khoroshilov A. V., Cheptsov V. Yu. CLOS Operating System Family. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 3, 2025., pp. 325-354. DOI: 10.15514/ISPRAS–2023–37(3)-23

1. Введение. История семейства операционных систем КЛОС

Первые публикации по операционной системе КЛОС (Кластерная Операционная Система) появились в 1977 году [1]. Основой публикации был новый подход к разработке операционных систем (ОС), который обобщал опыт, полученный в ИТМ и ВТ АН СССР в коллективе под руководством Л.Н. Королева и А.Н. Томилина, где создавалась операционная система для БЭСМ-6 Д-68, а потом НД-70 уже под руководством В.П. Иванникова.

Операционная система КЛОС ознаменовала собой переход на принципиально новый уровень постановки задачи разработки архитектуры операционной системы. В архитектуру КЛОС были включены решения, нацеленные на обеспечение выполнения не только таких традиционных требований к ОС как высокая производительность, малое время отклика, пространственное разделение пользовательских процессов, но и новых требований к архитектуре вычислительной системы, таких как взаимная изоляция программных компонентов ОС, минимизация объёмов программного кода, работающего в привилегированном режиме. В совокупности новые архитектурные решения позволили упростить развитие и перенос ОС на различные аппаратные платформы и расширить арсенал средств для обеспечения кибербезопасности, что стало особенно значимым в настоящее время.

Новый подход был реализован в проекте АС-6, в рамках которого создавались операционные системы для нескольких ЭВМ в локальной сети с единой распределенной системой управления внешними устройствами и линиями связи, распределенные информационные системы и распределенные прикладные системы управления космическими объектами. Общими чертами всех этих систем было наличие большого количества асинхронно выполняемых процессов, управляющих сложными внешними объектами и обменивающихся информацией между собой. Управление этими процессами, их синхронизация, организация взаимодействия между ними, обеспечение защиты, надежности и живучести системы в целом, миграция программ с одной платформы на

другую, повторное использование созданных ранее программ – вот основные проблемы, с которыми пришлось столкнуться в этой работе.

Созданием операционной системы для АС-6 руководил тогда еще молодой кандидат наук В. П. Иванников. Обобщение опыта разработки операционной системы АС-6 стало основой для докторской диссертации В.П. Иванникова на тему «Проблемы операционных систем многомашинных вычислительных комплексов и реализация операционных системы АС-6-БЭСМ-6» [2].

Название КЛОС – кластерная операционная система возникло по той причине, что в новом подходе предлагалось рассматривать ядро операционной системы как совокупность программных объектов изолированных друг от друга и взаимодействующих в соответствии со строгим протоколом – это важно для обеспечения изоляции и других значимых характеристик ОС. Аналогом такого подхода были первые объектно-ориентированные (ОО) системы. В 70-е годы наиболее продвинутой средой ОО программирования была система программирования на языке CLU, а базовые компоненты-объекты в CLU назывались кластерами. Впоследствии такие схемы построения ОС стали известны как микроядерные ОС.

После успешной реализации операционной системы для АС-6 (в частности, комплекс АС-6 отвечал за обработку траекторной информации совместного международного проекта «Союз-Аполлон» 1975 г.) концепция кластерной ОС была реализована для ЭВМ «Электроника ССБИС» и рабочей станции «Беста-88». Кроме того, была разработана и апробирована технология переноса базового уровня операционной системы с одной ЭВМ на другую. Этой теме была посвящена диссертация на соискание учёной степени кандидата физико-математических наук Г. В. Копытова «Принципы построения и реализация базового уровня кластерной операционной системы КЛОС» (1992) [3-6].

2. Современный период развития КЛОС

Новое поколение операционных систем на основе опыта разработки первых вариантов КЛОС появилось в результате работ по операционным системам реального времени для аэрокосмической техники. Эти работы были начаты сначала как инициативные исследования по развитию открытой реализации операционной системы РОК [7, 8], а затем были продолжены с такими индустриальными партнерами как ФАУ ГосНИИАС и АО РКС [9-11]. Результатом совместной работы с ГосНИИАС стала операционная система JetOS, которая в настоящее время проходит сертификацию на соответствие требованиям DO-178C/КТ-178C в составе авионики Суперджет Нью (SJ-100) и МС-21.

2.1 Классические и современные микроядерные ОС

Созданием специализированных ОС на сегодняшний день занимается значительное количество исследовательских групп в научных и промышленных организациях, что объясняется как многообразием аппаратных платформ, так и широким спектром задач, которые на них решаются. Концептуальные основы микроядерных ОС были сформулированы в конце 60-ых – начале 70-ых годов. За рубежом одним из первых опубликовал работы по новой архитектуре ОС (термина микроядро еще не существовало) датский ученый Бринч Хансен (Brinch Hansen). Базовые идеи состояли в том, что для обеспечения надежности нужно минимизировать объем кода, работающего в привилегированном режиме, компоненты ядра, выполняющие отдельные сервисы, должны быть изолированы друг от друга и взаимодействовать друг с другом не через общую память, а через специальные механизмы, например посредством передачи сообщений.

Создаваемые с 70-х годов за рубежом подобные микроядерным ОС, такие как Nucleus[12], несмотря на заложенные концепции, не давали ответа на вопрос границ применимости этого подхода, так как в первую очередь решали специализированные задачи на конкретном

оборудовании. Только с выходом микроядра Mach в 1985 году и Unix-совместимой ОС на его основе, стало возможным однозначно говорить не только о перспективности данного подхода, но и о его подтверждённой работоспособности. По мере развития микроядра Mach разработчики Р. Ф. Рашид и А. Теванян в коллективе соавторов опубликовали ряд статей[13-14], формулирующих основополагающие принципы построения микроядерных систем, которые применяются сегодня. В Mach были разделены понятия процесса и потока, обеспечивалась поддержка многоядерности, активно использовалась виртуальная память и механизмы «ленивого» копирования, была реализована система межпроцессного взаимодействия на основе контролируемых портов, являющаяся прообразом современного подхода системы безопасности на основе жетонов (capability-based security)[15].

Вскоре после появления Mach произошло осознание, что в рамках имеющихся аппаратных возможностей предложенный дизайн не обеспечивает достаточной производительности для решения типовых задач, включая сетевое взаимодействие или графический стек. Существуют три ключевых направления, которые позволили снять эти ограничения для современных микроядерных систем:

1. *Гибридизация микроядерного и монолитного подходов.* В зависимости от сферы применения и аппаратных возможностей отдельные подсистемы могут быть вынесены в привилегированный слой после проведения достаточного анализа безопасности и поверхности атаки.
   - Ядро XNU[16], как наиболее ранний представитель гибрида Mach и BSD, является практически монолитным, так как имеет в своём составе виртуальную файловую систему, сетевой стек и множество подсистем драйверов оборудования. Тем не менее, современный вектор развития данного ядра заключается в постепенном переносе компонентов, в особенности драйверов, в непривилегированный уровень.
   - Ядро HongMeng[17] является современным микроядром общего назначения, однако для обеспечения достаточной производительности реализует слой совместимости с Linux ABI и отдельные драйверы непосредственно в ядре, что является признаком применения гибридной архитектуры.
2. *Архитектурная переработка принципов межпроцессного взаимодействия в ядрах для минимизации накладных расходов[18]*. К таким изменениям можно отнести упрощение маршрутизации сообщений, синхронное переключение контекстов и механизмы уведомлений, отказ от глобальных блокировок уровня ядра, использование техник нулевого копирования при передаче сообщений. Данные архитектурные решения нашли применения в микроядрах, которые условно относят к микроядрам второго поколения и новее, например, L4 или seL4, как его современное развитие[19].
3. *Использование новых аппаратных возможностей[20]*. Усовершенствование подсистемы памяти, наличие контекстно-зависимых тегов внутри TLB, позволяющих избежать его полной очистки при смене адресных пространств, механизмы спекулятивного выполнения, аппаратно-поддержанные средства доступа к локальной памяти потока (TLS), аппаратная виртуализация и другие аппаратные нововведения позволили снизить накладные расходы для части механизмов микроядер.

Нельзя утверждать, что данные направления развития одинаково применимы к специализированным системам, однако, как видно из вышеизложенного, на современном этапе развития технологий сформировался ряд способов эффективной реализации микроядерной ОС общего назначения. Тем не менее, современные специализированные ОС, применяемые в промышленности, например, Genode или Helix, являются системами смешанной критичности и допускают совместный запуск ПО как общего, так и специального назначения для решения широкого набора задач. Такие системы сочетают в

себе разные подходы проектирования, и представленная в данной статье современная версия КЛОС развивается с учётом наиболее удачных мировых практик.

3. Основные архитектурные решения

Архитектура современной специализированной операционной системы должна быть надёжной, предсказуемой, безопасной и защищённой. Как правило, под этими терминами подразумевается микроядерная архитектура с жёстким реальным времем и пространственной изоляцией, требованием, которое часто игнорируется в системах реального времени, используемых в микроконтроллерах, например, FreeRTOS или RTEMS.

В основе архитектуры КЛОС лежат требования, предложенные в стандарте ARINC 653[21]. Данный стандарт определяет понятие надёжной изоляции (Robust Partitioning) и формулирует требования к операционной системы для обеспечения возможности работы независимых приложений разной степени критичности. Концепция надёжной изоляции изначально появилась в DO-248[22] и была дополнена в части многоядерности CAST-32A[23].

В ARINC 653 представлена реализация этой концепции, которая состоит из надёжной изоляции ресурсов (Robust Resource Partitioning) и надёжной временной изоляции (Robust Time Partitioning). Архитектура ARINC 653 декларирует, что каждое отдельное приложение в системе выделяется в отдельный *раздел (partition)*, который выполняется в изолированном адресном пространстве и использует заранее выделенные ему ресурсы. Разделы выполняются циклически последовательно в рамках заданных *временных окон (windows)*, определяемых текущим *расписанием (schedule)*. В данных терминах в документах изложен список требований для системы с надёжной изоляцией:

- Раздел не может повредить области памяти кода, данных или ввода-вывода других разделов.
- Раздел не может использовать больше разделяемых ресурсов, чем ему было выделено.
- Аппаратные сбои в одном разделе не могут приводить к негативным последствиям в других разделах.
- Раздел не может выполняться на процессорном ядре более длительное время, чем ему было выделено, вне зависимости от активности или неактивности других разделов на других процессорных ядрах.

Кроме требований на изоляцию, стандарт ARINC 653 определяет требования к интерфейсу ПО, однако в отличие от интерфейса, требования на изоляцию, как следует из CAST-32A, применимы и к другими специализированным системам. Рассмотрим более подробно особенности их реализации в КЛОС.

3.1 Архитектура пространственной изоляции

Архитектура ARINC 653 декларирует, что каждое отдельное приложение в системе выполняется в изолированном адресном пространстве, именуемом разделом. На рис. 1 представлена высокоуровневая часть архитектуры пространственной изоляции КЛОС. На рисунке изображены три раздела: два прикладных, один системный и структура микроядра ОСРВ, разделённая на аппаратно-зависимый и аппаратно-независимый слои.

В каждом разделе реализовано собственное функциональное ПО, использующее необходимый ему состав интерфейсов программирования. В разделах реализовано управление потоками (threads), которые в терминах ARINC 653 именуются процессами. Разделы взаимодействуют друг с другом и с операционной системой через сервисы, доступные непосредственно в адресном пространстве раздела и подкреплённые системными вызовами ядра ОС.

Базовый слой аппаратных абстракций выделен в отдельные компоненты ядра, в которые входит минимально необходимые средства: управление памятью, временем и прерываниями. Взаимодействие с периферийными устройствами выполняется вне ядра в системных разделах, которые, как и прикладные выполняются с пониженным уровнем привилегий.

Данная архитектура даже на таком уровне расширяет классические положения ARINC 653 в следующих аспектах:

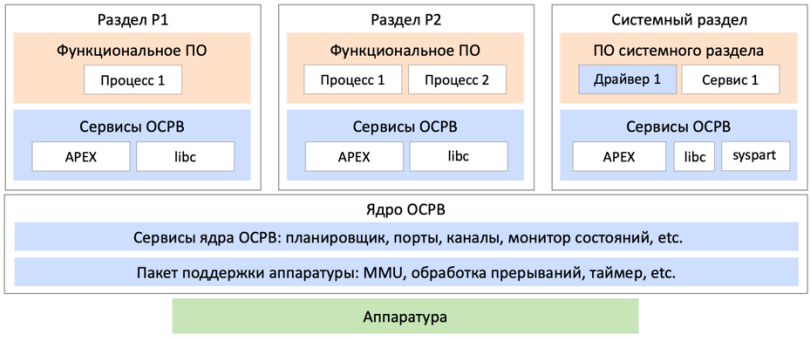

Рис. 1 Механизмы пространственной изоляции
Fig. 1. Robust space partitioning mechanisms

- Состав сервисов ОС, доступных прикладному функциональному ПО в отдельно взятом разделе, конфигурируется как в большую, так и в меньшую сторону. Разделам могут быть доступны сервисы APEX, описанные в стандарте ARINC 653, стандартные библиотеки языков программирования (например, Ada, C, C++, Modula-2), отраслевые расширения (например, сервисы для работы в рамках распределённых бортовых сетей, сервисы управления робототехникой, средства организации доверенных вычислений TEE), сервисы для проведения модульного и интеграционного тестирования и др. Адаптация нового интерфейса программирования для одного из разделов, например, элементов стандарта POSIX, не влечёт за собой изменений в интерфейсе программирования для других разделов, позволяя проектировать гетерогенные системы смешанной критичности.
- Изоляция периферийных устройств в отдельные системные разделы (см. рис. 2) позволяет изолировать целые подсистемы ОС, например, файловую систему или сетевой стек в отдельные адресные пространства, уменьшить поверхность атаки и сделать вычислительную среду более гранулярной. Количество системных разделов и распределение компонентов взаимодействия с периферийными устройствами по данным системным разделам определяется системным интегратором в зависимости от аппаратных возможностей целевой аппаратуры. Таким образом пространственная изоляция возможна не только между прикладным и системным ПО, а также системным ПО и ядром ОС, но и между отдельными компонентами системного ПО.

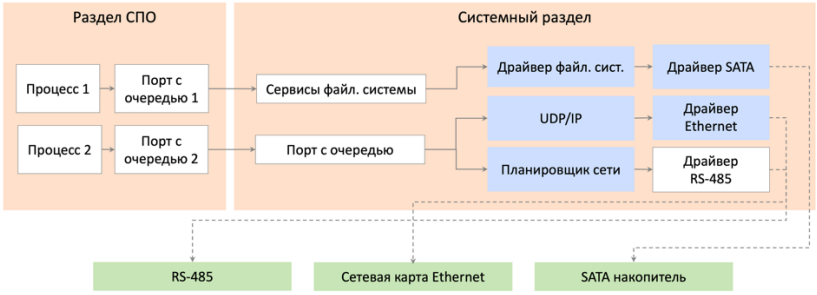

Рис. 2 Механизмы изоляции периферийных устройств
Fig. 2. Robust hardware peripheral partitioning mechanisms

За счёт активного использования устройства управления памятью (MMU) в КЛОС разделам предоставляются три основных метода передачи сообщений:

1. Через стандартные средства ARINC 653, такие как порты с очередью, порты без очереди и SAP. Данный механизм передачи сообщений гарантирует безопасность по памяти и обеспечивает доставку установленного количества сообщений. К его минусам можно отнести два копирования (в момент отправки и в момент получения) и накладные расходы на системные вызовы.
2. Через механизм блоков памяти, который в КЛОС расширяет семантику ARINC 653, введением участков общей памяти между участниками обмена. Права доступа и политики кэширования к блокам памяти настраиваются, что позволяет проектировать разные схемы обмена, в том числе с реализацией гарантий безопасности по памяти и обеспечением нулевого количества копирований.
3. Через механизм удалённых процедур в разделе. В этом случае происходит синхронная передача управления в обработчик в другом адресном пространстве внутри временного окна текущего раздела. Механизм фактически позволяет реализовывать системные вызовов в пользовательском пространстве одного из разделов. Количество параллельных обработчиков настраивается для предоставления временных гарантий при множественном доступе. Возможна передача и возврат скалярных значений.

Управление MMU в КЛОС выполняется по заранее вычисленной конфигурации, что минимизирует накладные расходы на использование виртуальной памяти и позволяет проводить статическую и динамическую верификацию пространственной изоляции инструментальными средствами. Данный подход, одновременно являясь простым и гибким, наделяет подсистему памяти свойством равномерно высокой производительности как в среднем, так и в наихудшем случае. Более подробно данные особенности изложены в отдельных работах[24-25].

3.2 Архитектура временной изоляции

В основу архитектуры ARINC 653 положена система статически заданного расписания, ограничивающее время работы каждого раздела в рамках одного основного временного кадра (периода расписания) заданными временными промежутками — окнами.

На рис. 3 представлена верхнеуровневая архитектура временной изоляции КЛОС. Часть, описывающая работу расписаний модуля 1, представляет собой классическую реализацию расписаний современного стандарта ARINC 653 (2024 года) с поддержкой многоядерности.

Двум разделам P1 и P2 назначены окна по 2 мс каждое с общим основным временным кадром в 4 мс. Разделу P1 назначено одно процессорное ядро, а раздел P2 работает с двумя процессорными ядрами (0-е и 1-е) в режиме симметричной многопроцессорности (SMP), для обеспечения гарантий временной изоляции во время работы раздела P1 1-е ядро процессора простаивает (IDLE). Так как ARINC 653 допускает наличие нескольких таких

заранее заданных расписаний, по окончанию основного временного кадра может происходить переключение между расписаниями по инициативе разделов, имеющих специальные привилегии.

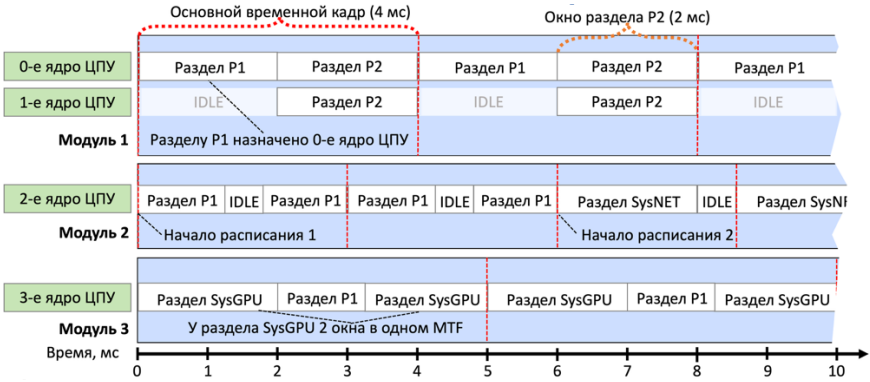

Рис. 3 Механизмы временной изоляции

Fig. 3. Robust time partitioning mechanisms

Данный подход предоставляет сильные гарантии временной изоляции между приложениями, однако он часто критикуется за недостаточную реактивность. Для решения данной проблемы в КЛОС предложены следующие расширения ARINC 653:

- Количество окон у одного раздела не ограничено и может располагаться в произвольной последовательности. Это позволяет повысить производительность ввода-вывода, создавая цепочки окон вида прикладной раздел → системный раздел → прикладной раздел.
- Платформы, поддерживающие несколько ядер процессора, могут быть задействованы в режиме асимметричной многоядерности (AMP). На каждой группе процессорных ядер, состав которых определяется системным интегратором, запускается отдельный модуль (экземпляр) ОС с независимым набором разделов и расписаний. Память между модулями за исключением блоков общей памяти, используемых для обмена сообщениями, не является когерентной, что позволяет минимизировать влияние модулей на временные характеристики друг друга (см. рис. 4).
- Разделы, имеющие специальные привилегии, могут делегировать своё процессорное время для работы других разделов, например, через механизм удалённых процедур.

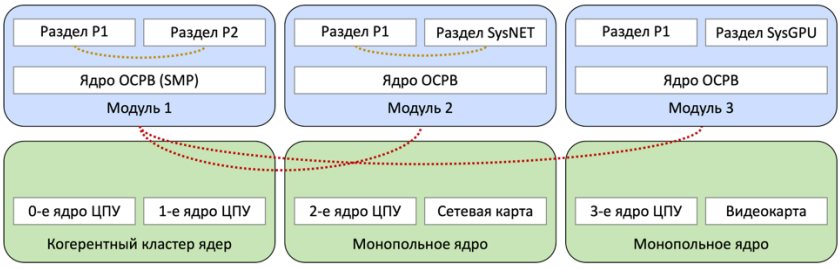

Рис. 4 Механизмы временной изоляции в сценарии многоядерности

Fig. 4. Robust time partitioning mechanisms in multicore scenario

На техническом уровне в КЛОС реализован ряд решений, направленный на повышение общей отзывчивости системы, например, в части реализаци системных вызовов, и минимизации времени работы в наихудшем случае, например, в части реализации планировщика, работающего в непериодическом режиме с квантованием времени.

Применение данного подхода позволяет полностью исключить срабатывание «холостых» прерываний от таймера, снизить время кванта в сравнении с периодическим таймером, при этом сохранив латентность на предсказуемом уровне за счёт корректировки кванта планировщика под задачу. Более подробно данные особенности изложены в отдельной работе[26].

3.3 Основные архитектурные гарантии

Резюмируя вышеизложенное, можно переформулировать требования стандарта ARINC 653 и сопутствующих документов в виде свойств, которые предоставляет КЛОС:

- Доступные разделу или ядру ОСРВ ресурсы, включая области памяти и время выполнения, должны определяться на основании конфигурации, заданной статически до запуска ОСРВ.
- Привилегии доступа к произвольному участку памяти должны соответствовать минимально необходимым для нормального функционирования раздела или ядра ОСРВ.
- Выполнение раздела на процессорном ядре должно быть ограничено в соответствии с конфигурацией, и не может быть превышено.
- Выполнение раздела на процессорном ядре не должно прерываться, кроме как для выполнения переключения между разделами или выполнения действий, затребованных самим разделом.
- Время переключения между разделами должно быть минимальным.

Нетрудно заметить, что эти свойства одновременно гарантируют более сильные гарантии по изоляции в сравнении с требованиями ARINC 653, но при этом предоставляют дополнительные гарантии в части обеспечения высокой производительности и адаптивности системы к разным сценариям использования.

Проведённые исследования позволили продемонстрировать, что выработанная архитектура может одинаково хорошо адаптироваться как к близким для ARINC 653 сферам, например, к сфере автоматических космических аппаратов[27], так и к не относящимся к ARINC 653 сценариям использования, например, Trusted Execution Environment или окружение для тестирования ПО[28].

4. Поддерживаемые аппаратные платформы

ОС с хорошей переносимостью между аппаратными платформами характеризуется следующими свойствами:

1. Наличие независимого слоя абстракций аппаратуры (Hardware Abstraction Layer) с заданным интерфейсом. Адаптация ОС к новой аппаратной платформе не требует изменения аппаратно-независимых компонентов и касается только компонентов, относящихся к пакету поддержки платформы, и инструментов системы сборки.
2. Эффективное использование ресурсов поддерживаемых вычислительных платформ. В том числе особенностей подсистемы памяти и аппаратных потоков (процессорных ядер).
3. Способность масштабироваться к разным объёмам ресурсов и разной вычислительной мощности. В том числе к разным объёмам ОЗУ и ПЗУ, к широкому диапазону рабочих частот ЦПУ.
4. Высокая переносимость прикладного кода как на уровне исходных кодов, так и на бинарном уровне.

Так как в различных сегментах промышленности сложились свои привычные к использованию аппаратные платформы, а отрасль встраиваемых систем в целом достаточна

гибкая в части выбора целевой аппаратуры, КЛОС изначально проектировалась для обеспечения хорошей переносимости между аппаратными платформами.

Пакет поддержки аппаратуры КЛОС (Platform Support Package, PSP) состоит из трёх гибко конфигурируемых уровней абстракций (см. рис. 5):

- *Пакет поддержки архитектуры* (Architecture Support Package, ASP) представляет собой часть PSP, общую для заданной архитектуры набора команд (Instruction Set Architecture, ISA). Например, AArch64, MIPS32, PowerPC32, RISC-V RV32 и др.
- *Пакет поддержки семейства целевых аппаратных платформ* (Common Board Support Package, BSP Common) представляет собой часть PSP, которая специализирует реализацию архитектуры набора команд отдельным производителем. Например, Cortex-A55, PowerPC e500, КОМДИВ и др.
- *Пакет поддержки целевой аппаратной платформы* (Board Support Package, BSP) представляет собой часть PSP, которая специализирует отдельную целевую аппаратную платформу. Например, отладочная плата ВК018 или МВ115. Данный уровень передаётся пользователю ОС для дальнейшей специализации с целью разработки конечных устройств.

.

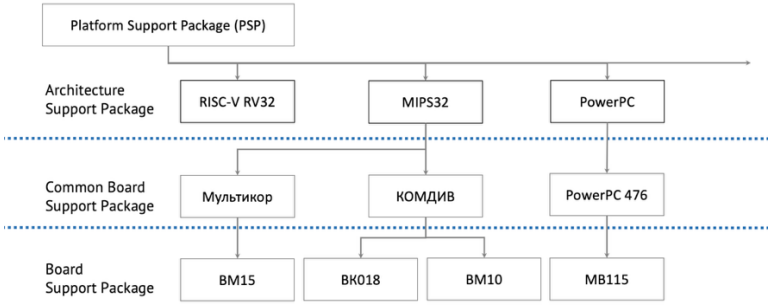

Рис. 5 Структура пакета поддержки аппаратуры
Fig. 5. Structure of the hardware support package

Данные уровни не являются монолитными и строятся на основе переиспользуемых компонентов, которые могут быть общими даже для разных ASP, например, когда периферийные IP-блоки одного производителя внедрялись в устройства с разными ISA.

Ядро КЛОС активно использует атомарные операции и модель памяти языка С версии 2011 года, за счёт чего хорошо адаптирована к архитектурам со «слабой» моделью памяти. Для повышения скорости и предсказуемости времени работы подсистемы памяти на поддерживаемых ISA активированы режимы strict alignment, гарантирующие выброс исключения процессора при разыменовании невыровненных указателей. Механизмы поддержки многоядерности в КЛОС настраиваемы, что позволяет получать высокую производительность как на одноядерных, так и на многоядерных платформах, а также работать на многоядерных платформах в оптимизированном одноядерном режиме.

Отдельное внимание в КЛОС уделяется проблеме фактической переносимости кода между целевыми платформами. В большинстве встраиваемых систем для простоты используется ABI на основе System V или Linux. Это позволяет переиспользовать существующие инструменты разработки (компиляторы, компоновщики, отладчики и пр.) между разными системами. Тем не менее, по историческим причинам между разными целевыми платформами в реализации данных ABI, например, на языке С, существуют различия даже для ISA одной разрядности.

КЛОС не является исключением и переиспользует System V ABI для организации интерфейсов прикладного и системного программистов, однако в дополнении к этому

вводит *единую систему размеров базовых типов* для 32-битных и 64-битных платформ (см. Табл. 1). Данный подход позволяет сделать потребление ресурсов, в том числе стека, более предсказуемым при переходе между компиляторами или с одной ISA на другую.

Таблица 1. Размеры базовых типов

Table 1. Dimensions of basic types

Тип	32-битные платформы		64-битные платформы	
	Размер	Выравнивание	Размер	Выравнивание
bool, _Bool	1 байт	1 байт	1 байт	1 байт
signed char	1 байт	1 байт	1 байт	1 байт
signed short	2 байта	2 байта	2 байта	2 байта
signed int	4 байта	4 байта	4 байта	4 байта
signed long	4 байта	4 байта	8 байтов	8 байтов
signed long long	8 байтов	8 байтов	8 байтов	8 байтов
_BitInt(128)	16 байтов	16 байтов	16 байтов	16 байтов
unsigned char	1 байт	1 байт	1 байт	1 байт
unsigned short	2 байта	2 байта	2 байта	2 байта
unsigned int	4 байта	4 байта	4 байта	4 байта
unsigned long	4 байта	4 байта	8 байтов	8 байтов
unsigned long long	8 байтов	8 байтов	8 байтов	8 байтов
unsigned _BitInt(128)	16 байтов	16 байтов	16 байтов	16 байтов
enum	4 байта	4 байта	4 байта	4 байта
void *	4 байта	4 байта	8 байтов	8 байтов
void (*)(void)	4 байта	4 байта	8 байтов	8 байтов
float	4 байта	4 байта	4 байта	4 байта
double	8 байтов	8 байтов	8 байтов	8 байтов
long double	8 байтов	8 байтов	8 байтов	8 байтов

Кроме этого, фиксируется также и реализация базовых типов для типов фиксированной ширины (см. Табл. 2) и определяется семантика работы отдельных типов данных. Так перечисляемому тип enum гарантируется 32-битная размерность, float и double реализуются в соответствии со стандартом IEEE 754 (с учётом ограничений аппаратуры в части корректности округления и вычисления значений), для атомарных типов не более разрядности целевой платформы гарантируется работа без блокировок.

Таблица 2. Реализация базовых типов

Table 2. Implementation of base types

Тип	32-битные платформы	64-битные платформы
int8_t	signed char	signed char
int16_t	signed short	signed short
int32_t	signed int	signed int
int64_t	signed long long	signed long
uint8_t	unsigned char	unsigned char
uint16_t	unsigned short	unsigned short
uint32_t	unsigned int	unsigned int
uint64_t	unsigned long long	unsigned long
intmax_t	signed long long	signed long
uintmax_t	unsigned long long	unsigned long long
ssize_t	signed int	signed long
size_t	unsigned int	unsigned long
ptrdiff_t	signed int	signed long
intptr_t	signed int	signed long
uintptr_t	unsigned int	unsigned long
wchar_t	signed int	signed int
wint_t	unsigned int	unsigned int
enum	unsigned int	unsigned int

КЛОС поддерживает как платформы с порядком байтов от младшего к старшему (Little Endian), так и платформы с порядком байтов от старшего к младшему (Big Endian). Переход между такими платформами для прикладного программиста нередко сопровождается дополнительными затратами времени на поиск ошибок переносимости. Для решения этой

проблемы в КЛОС было уделено дополнительное внимание к процессорам, традиционно использующим Big Endian порядок байтов, таким как КОМДИВ и вариации PowerPC. Как выяснилось, хотя такое решение не является распространённым, для большинства из них поддерживается смешенный порядок байтов. После некоторой переработки пакета поддержки аппаратуры КЛОС смогла работать на всех поддерживаемых ASP с Little Endian порядком байтов без потери производительности, включая вариации PowerPC 15-летней давности и старее: ядра e500v2, e500mc, 476FP.

На момент написания данной статьи КЛОС поддерживает следующие аппаратные платформы:

- AArch64 (Cortex-A53, Cortex-A55), например, процессоры семейства RK35xx;
- ARM (Cortex-A7, Cortex-A9, Cortex-M4), например, процессоры i.MX6 или STM32F4;
- PowerPC (e500mc, e500v2, 476FP), например, процессоры p1010, p3041, 1888TX018;
- MIPS (MIPS Release 1, MIPS Release 2 / MIPS32, КОМДИВ, Мультикор), например, процессоры 1892ВМ15АФ и К5500ВК018;
- RISC-V (RV32 IMA, Syntacore SCR5);
- x86 (Intel Prescott и новее).

Для данных платформ реализованы следующие подсистемы сопряжения с аппаратурой: eMMC, Ethernet, i2c, NVRAM (FRAM, EEPROM), ONFI NAND, Parallel и SPI NOR, PCI, RTC, SATA, SpaceWire, SPI, VirtIO, ГОСТ Р 52070-2003 (МКИО), UART, SPI NOR и др.

5. Инструментарий разработчика

В связи с тем, что отдельные ОС на базе КЛОС применяются в сферах с особыми требованиями к безопасности и подлежат обязательной сертификации на соответствие таким международным стандартам как КТ-178С, к прикладному ПО, работающему под управлением КЛОС, часто также предъявляются аналогичные требования по соблюдению строгих требований безопасности и надёжности. В отличие от отечественного сегмента, где данное требование является локальным для отдельных сфер промышленности (например, в поле действия КТ-178С), современные зарубежные ведомства требуют[29] архитектурных гарантий по недопущению ошибок, связанных с работой с памятью, во всём ПО критической инфраструктуры.

В следствие этого, среда разработчика КЛОС изначально ориентирована на разработку безопасных приложений с предсказуемым поведением и подкреплена интерфейсными и инструментальными средствами. Для борьбы с ошибками работы с памятью и гонками в КЛОС доступны следующие средства:

- Языки программирования с безопасной работой с памятью. В КЛОС поддерживается разработка приложений на языке Ada с необходимыми биндингами для сервисов ARINC 653 и ОС.
- Статические средства выявления ошибок. На уровне системы сборки КЛОС произведена интеграция различных средств статического анализа начального уровня. Например, средств линтинга, таких как как Clang Tidy совместно с Clang Format и средств статического анализа на уровне одного модуля, таких как Clang Static Analyzer. Кроме этого, для интерфейсов КЛОС были успешно разработаны спецификации для промышленных средств статического анализа, таких как Svace[30], что позволяет максимизировать эффективность срабатываний.
- Динамические средства выявления ошибок. В КЛОС реализованы все основные санитайзеры LLVM: Address, Memory, Thread, Undefined Behavior, а также собственные инструменты, такие как RaceHunter[31-33].
- Встроенная система тестирования. В системе тестирования КЛОС реализованы адаптеры для проведения модульного, интеграционного и полносистемного

тестирования. Доступны средства профилирования кода и сбора покрытия по строкам, ветвям, а также по критерию MC/DC.

- Расширенный пакет полносистемной эмуляции аппаратуры на основе QEMU с поддержкой единой схемы отладки кода как на эмулируемых, так и на аппаратных платформах (см. рис. 6). В эмуляторе поддерживаются: режим детерминированной симуляции и обратная отладка для многократного воспроизведения и исследования возникших ошибок. При необходимости обеспечивается сопряжение физических устройств к эмулируемому ПО (например, по интерфейсам RS-485 или Ethernet).

Кроме этого, ведутся исследования по адаптации и снижению порога входа для средств автоматической статической верификации в контексте верификации прикладного ПО на базе КЛОС[34].

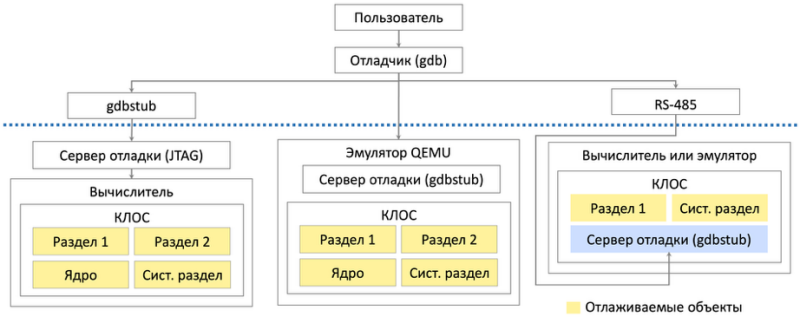

Рис. 6 Структура пакета поддержки аппаратуры

Fig. 6. Structure of the hardware support package

6. Заключение

Семейство операционных систем КЛОС появилось ещё в начале 70-х годов прошлого века на основе опыта разработок операционных систем научной группой академика В. П. Иванникова. За прошедшие годы как в области микроядерных операционных систем, так и в спектре задач, которыми занимаются разработчики операционных систем в ИСП РАН, многое изменилось. Важнейшие принципы микроядерных ОС остались неизменными – проектные решения должны обеспечивать минимизацию программ, работающих в привилегированном режиме, и быть в максимальной степени изолированы друг от друга. Это даёт качественное повышение надёжности и защищенности ОС, а также облегчает портирование ОС и приложений на разные аппаратные платформы. В настоящее время в рамках семейства активно развиваются ОС для аэрокосмической отрасли. Во многом эти версии похожи между собой, но между ними есть и серьёзные различия. Так ОСРВ для сценариев полностью автоматической эксплуатации предусматривает возможности удалённой коррекции и перезагрузки, что необходимо в ходе длительных полетов.

В силу того, что все современные ОС семейства КЛОС предназначены для систем ответственного назначения, в процессы их разработки включены самые передовые технологии поддержки Разработки Безопасного ПО (РБПО), а также специализированные средства отладки, отработки, тестирования и верификации операционных систем и базового слоя программно-аппаратных комплексов. В сочетании с отечественными аппаратными платформами операционные системы семейства КЛОС предоставляют доверенную платформу для создания программно-аппаратных комплексов широкого спектра назначений.

Благодарности

Авторы выражают глубокую благодарность многочисленным участникам проектов КЛОС, начиная с 70-х годов XX века вплоть до наших дней.

Список литературы / References

1. В. П. Иванников. «Использование кластеров в операционной системе» // ДАН СССР, 1977, Т. 237, № 2, стр. 2800–2833.
2. В. П. Иванников. Проблемы операционных систем многомашинных вычислительных комплексов и реализация операционных системы АС-6-БЭСМ-6 [Текст] : Автореф. дис. на соиск. учен. степ. д. ф.-м. н. : 01.01.10, 1979.
3. И. Б. Бурдонов, В. П. Иванников, А. С. Косачев, Г. В. Копытов, С. Д. Кузнецов. Проект КЛОС: к объектно-ориентированной среде разработки прикладных систем.- Управляющие машины и системы, 1992, N 1/2, стр. 61-65/
4. И. Б. Бурдонов, Г. В. Копытов, А. С. Косачев, С. Д. Кузнецов, Ю. П. Смирнов, В. Н. Юдин, КЛОС: операционная система и технология программирования. Сб. «Вопросы кибернетики. Программное обеспечение высокопроизводительной системы». Под ред. В. П. Иванникова. М., НСК АН СССР, 1986 г., стр.34-57.
5. И. Б. Бурдонов, В.П. Иванников В.П., А.С. Косачев. Проект КЛАСТОС.- Труды SORUCOM-2011, Вторая международная конференция «Развитие вычислительной техники и ее ПО в России и странах бывшего СССР». Великий Новгород, 12-16 сентября 2011 г., стр. 76-82.
6. Г. В. Копытов. «Принципы построения и реализация базового уровня кластерной операционной системы КЛОС» (диссертация на соискание учёной степени кандидата физико-математических наук по специальности 05.13.11). 1992 г. https://search.rsl.ru/ru/record/01002669502 (доступ 06.11.2025).
7. Н. В. Пакулин, А. В. Хорошилов. “Свободная реализация ARINC-653-совместимой операционной системы реального времени”. Сборник докладов Двенадцатой конференции разработчиков свободных программ, сс. 6-7, Калуга, 16-18 октября 2015 г. Альт Линукс, Москва, 2015, ISBN 978-5-905167-19-5.
8. POK, a real-time kernel for secure embedded systems // https://pok-kernel.github.io/
9. К. М. Маллачиев, Н. В. Пакулин, А. В. Хорошилов. “Устройство и архитектура операционной системы реального времени”. Труды Института Системного Программирования РАН, Том 28-2, 2016, с. 181-192. DOI: 10.15514/ISPRAS-2016-28(2)-12
10. K. A. Mallachiev, N. V. Pakulin, A. V. Khoroshilov, D. V. Buzdalov, “Using modularization in embedded OS”, Труды ИСП РАН, 29:4 (2017), 283–294. 3.
11. Н. В. Пакулин. Микроядерная сертифицируемая операционная система реального времени // Конференция OSDAY-2017, https://osday.ru/2017/presentations/pakulin/pakulin.pdf
12. B. Hansen. The nucleus of a multiprogramming system. Communications of the ACM, 13(4), 1970, 238–241. DOI: 10.1145/362258.362278
13. Michael J. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, Michael Young. Mach: A new kernel foundation for UNIX development. USENIX Summer Conference, 1986.
14. David Black, David B. Golub, Daniel P. Julin, Richard F. Rashid, Richard P. Draves, Randall W. Dean, Alessandro Forin, Joseph Barrera, Hideyuki Tokuda, Gerald R. Malan, David Bohman. Microkernel operating system architecture and Mach. Journal of information processing, 14(4), 442-453, 1991.
15. Е. С. Басков. Использование capability-based security в ядрах операционных систем // Конференция OSDAY-2025, https://osday.ru/downloads/Baskov.pdf
16. Amit Singh. Mac OS X Internals: A Systems Approach. Addison-Wesley Professional, 2006. ISBN 0132702266, 9780132702263. 1680 с.
17. Haibo Chen, Huawei Central Software Institute and Shanghai Jiao Tong University; Xie Miao, Ning Jia, Nan Wang, Yu Li, Nian Liu, Yutao Liu, Fei Wang, Qiang Huang, Kun Li, Hongyang Yang, Hui Wang, Jie Yin, Yu Peng, and Fengwei Xu, Huawei Central Software Institute. Microkernel Goes General: Performance and Compatibility in the HongMeng Production

Microkernel. 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24). 465-485, 2024
18. H. Härtig, M. Hohmuth, J. Liedtke, S. Schönberg, J. Wolter. The Performance of µ-Kernel-Based Systems. 16th ACM Symposium on Operating Systems Principles (SOSP ’97). 1997
19. G. Heiser. The seL4 Microkernel – An Introduction. Revision 1.5. The seL4 Foundation. 2025. // https://sel4.systems/About/seL4-whitepaper.pdf
20. G. Heiser and K. Elphinstone. L4 Microkernels: The Lessons from 20 Years of Research and Deployment. ACM Trans. Comput. Syst. 34, 1, Article 1. 2016, 29 pages. DOI: 10.1145/2893177
21. Avionics application software standard interface part 0 overview of ARINC 653, ARINC specification 653P0-3, 2021.
22. DO-248C. Supporting Information for DO-178C and DO-278A. RTCA: SC-205, 2011. — 166 p.
23. CAST-32A. Multi-core Processors. [Текст]. — The Federal Aviation Administration: Certification Authorities Software Team, 2016. — 23 p.
24. V. Cheptsov and A. Khoroshilov, "Robust Resource Partitioning Approach for ARINC 653 RTOS," 2023 Ivannikov Ispras Open Conference (ISPRAS), Moscow, Russian Federation, 2023, pp. 33-39, doi: 10.1109/ISPRAS60948.2023.10508165.
25. С. А. Зеленова Статическое распределение памяти для операционных систем реального времени // Труды ИСП РАН. 2024. №3. URL: https://cyberleninka.ru/article/n/staticheskoe-raspredelenie-pamyati-dlya-operatsionnyh-sistem-realnogo-vremeni (дата обращения: 01.11.2025).
26. В. Ю. Чепцов. Метод надёжной временной изоляции для ARINC 653 ОСРВ // Конференция OSDAY-2024 // https://0x1.tv/20240620I (дата обращения: 01.11.2025).
27. В. Ю. Чепцов. А. В. Хорошилов. Проектирование бортовой ОСРВ с жёстким реальным временем для космического применения // Конференция OSDAY-2023 https://osday.ru/2023/presentations/cheptsov.pdf (дата обращения: 01.11.2025).
28. А. И. Аветисян et al. TestOS: окружение для тестирования ПО. Сборник технологий ИСП РАН. 2024 г. // https://www.ispras.ru/technologies/testos/
29. CISA, NSA. Memory Safe Languages: Reducing Vulnerabilities in Modern Software Development // https://media.defense.gov/2025/Jun/23/2003742198/-1/-1/0/CSI_MEMORY_SAFE_LANGUAGES_REDUCING_VULNERABILITIES_IN_MODERN_SOFTWARE_DEVELOPMENT.PDF (дата обращения: 01.11.2025).
30. А. И. Аветисян, А. Е. Бородин. Механизмы расширения системы статического анализа Svace детекторами новых видов уязвимостей и критических ошибок. Труды Института системного программирования РАН. 2011;21.
31. V. Cheptsov, A. Khoroshilov, "Dynamic Analysis of ARINC 653 RTOS with LLVM," 2018 Ivannikov Ispras Open Conference (ISPRAS), Moscow, Russia, 2018, pp. 9-15, DOI: 10.1109/ISPRAS.2018.00009.
32. Е. Ельчинов. Адаптация алгоритма ThreadSanitizer для обнаружения гонок по данным в ядре ОСРВ. Международная конференция «Иванниковские чтения». 2025 г. // https://ivannikov-ws.org
33. Е. А. Герлиц. Инструмент для поиска гонок по данным RaceHunter. Труды Института системного программирования РАН. 2023;35(6):135-156. DOI: 10.15514/ISPRAS-2023-35(6)-8
34. S. L. Lesovoy. Extracting architectural information from source code of ARINC 653-compatible application software using CEGAR-based approach. Труды ИСП РАН, 30:3 (2018), 31–46. DOI: 10.15514/ISPRAS-2018-30(3)-3

Информация об авторах / Information about authors

Игорь Борисович БУРДОНОВ – доктор физико-математических наук, главный научный сотрудник ИСП РАН. Научные интересы: формальные спецификации, генерация тестов, технология компиляции, системы реального времени, операционные системы, объектно-ориентированное программирование, сетевые протоколы, процессы разработки программного обеспечения.

Igor Borisovich BURDONOV – Dr. Sci. (Phys.-Math.), Chief Researcher at ISP RAS. Research interests: formal specifications, test generation, compilation technology, real-time systems, operating systems, object-oriented programming, network protocols, software development processes.

Александр Сергеевич КОСАЧЕВ – кандидат физико-математических наук, ведущий научный сотрудник ИСП РАН. Научные интересы: формальные спецификации, генерация тестов, технология компиляции, системы реального времени, операционные системы, объектно-ориентированное программирование, сетевые протоколы, процессы разработки программного обеспечения.

Alexander Sergeevitch KOSSATCHEV – Cand. Sci. (Phys.-Math.), a Leading Researcher at ISP RAS. Research interests: formal specifications, test generation, compilation technology, real-time systems, operating systems, object-oriented programming, network protocols, software development processes.

Александр Константинович ПЕТРЕНКО — профессор, д.ф.-м.н., заведующий отделом Технологий программирования ИСП РАН, профессор кафедр Системного программирования ВМК МГУ и ФКН НИУ ВШЭ. Научные интересы: формальные методы программной инженерии, операционные системы, языки спецификаций и моделирования, верификация.

Alexander Konstantinovich PETRENKO — Professor, Dr. Sci. (Phys.-Math.), Head of the Software Engineering Department at the Ivannikov Institute for System Programming, Russian Academy of Sciences, Professor of MSU and the Faculty of Computer Science, NRU HSE. Research interests: formal methods of software engineering, operating systems, specification and modeling languages, verification.

Алексей Владимирович ХОРОШИЛОВ – кандидат физико-математических наук, ведущий научный сотрудник, руководитель Центра верификации ОС Linux в ИСП РАН, доцент кафедр системного программирования МГУ, ВШЭ и МФТИ. Основные научные интересы: методы проектирования и разработки ответственных систем, формальные методы программной инженерии, методы верификации и валидации, тестирование на основе моделей, методы анализа требований, операционная система Linux.

Alexey Vladimirovich KHOROSHILOV – Cand. Sci. (Phys.-Math.), Leading Researcher, Director of the Linux OS Verification Center at ISP RAS, Associate Professor of System Programming Departments at MSU, NRU HSE, and MIPT. Main research interests: design and development methods for critical systems, formal methods of software engineering, verification and validation methods, model-based testing, requirements analysis methods, Linux operating system.

Виталий Юрьевич ЧЕПЦОВ – архитектор операционных систем в ИСП РАН. Основные научные интересы: встраиваемыми системы ответственного назначения, системы с жёстким реальным временем для бортовой аппаратуры, безопасность Unix-совместимых ОС общего назначения, загрузочное программное обеспечение UEFI.

Vitaliy Yurievich CHEPTSOV – OS architect at ISP RAS. Main research interests: safety-critical embedded systems, hard real-time systems for airborne equipment, security of general-purpose Unix-like operating systems, UEFI boot software.