**DOI: 10.15514/ISPRAS-2019-1(2)-1**

# Статическая раскладка памяти как средство организации MMIO и DMA в ОСРВ с пространственной изоляцией

[1] ----------, *ORCID: 0000-0000-0000-0000* <---------->
[1] ----------.

**Аннотация.** Задача статической раскладки памяти важна в контексте обеспечения безопасности и надёжности ОСРВ. Архитектурные особенности аппаратных платформ на первый взгляд создают целый зоопарк дополнительных условий, тем самым усложняя задачу статической раскладки памяти. Построение общей теории позволяет создать строгие рамки и вписать различные условия в общий план решения. В работе предлагается анализ ряда нетривиальных архитектурных особенностей (таких, как фиксированная трансляция адресов, DMA-операции, арифметическая адресация устройств и др.), связанных с выделением областей памяти со специфическими ограничениями на отображения, строится общая система понятий для таких областей, формулируются задачи распределения памяти и демонстрируются пути их решения, пригодные к промышленной разработке. Подход был успешно апробирован в ОСРВ КЛОС, разрабатываемой в ИСП РАН.

**Ключевые слова:** операционные системы реального времени; статическая раскладка памяти; автоматизация раскладки памяти; фиксированная трансляция адресов; DMA-операции; арифметическая адресация устройств.

**Для цитирования:** ----------.

# Static memory allocation for memory areas with mapping constraints

[1] ---------- *ORCID: 0000-0000-0000-0000* <---------->
[1] ----------.

**Abstract.** Static memory layout is important in the context of ensuring the safety and reliability of RTOS. At first glance, architectural features constitute the real zoo of additional conditions, thereby complicating the problem of static memory layout. A general theory allows to create a strict framework and fit various conditions into the unified solution plan. In the paper we analyze some of non-trivial architectural features (such as fixed address mapping, DMA operations, address arithmetic for devices, etc.) related to memory areas with specific mapping constraints, build a general concepts for such areas, formulate memory allocation problems and demonstrate ways to solve them. suitable for industrial development. The approach has been successfully tested in the RTOS, which is being developed at the ISP RAS.

**Keywords:** real-time operating system; static memory allocation; memory allocation methods; robust partitioning; fixed address mapping; DMA-operations; address arithmetic.

**For citation:** ----------

**Acknowledgements.** Блок «Благодарности» на английском языке.

## *1. Введение*

Пространственная изоляция процессов является одним из средств обеспечения надёжности и безопасности операционных систем реального времени. Важность соблюдения этого требования подчёркивается внесением его в отраслевые стандарты, такие как ARINC 653 [1][2].

Точное распределение памяти для различных нужд до запуска системы позволяет не только обеспечить пространственную изоляцию, но и добиться снижения накладных расходов при использовании механизмов переключения виртуальных адресных пространств. Таким образом, развитие методов статического распределения памяти приобретает особое значение в контексте создания ОСРВ [3][4][5][6].

Наличие общей теории устройства разных архитектур с точки зрения построения статической раскладки памяти является мощным подспорьем при создании рабочих инструментов. Система понятий, объединяющих особенности архитектур, позволяет выделять задачи общего плана, давать им строгие формулировки и привлекать для их решения математический аппарат.

В данной статье предпринимается попытка обобщения и моделирования некоторых аппаратных особенностей отдельных платформ для более эффективного построения инструментов статической раскладки памяти. В разделе 2 приводятся некоторые общие сведения об устройстве требований на память и статической раскладке памяти. Раздел 3 посвящён обзору практических сценариев. В разделе 4 на основе приведённых сценариев строится общая модель. Раздел 5 содержит строгие формулировки задач распределения памяти в терминах построенной модели и краткое описание возможных подходов к их решению.

## *2. Статическая раскладка памяти*

Напомним термины, связанные со статической раскладкой памяти.

*Статической раскладкой памяти* называется распределение памяти между программными компонентами, строящееся до запуска системы и удовлетворяющее заданным заранее требованиям.

Требования на память исходят из разных источников (аппаратные особенности, запросы операционной системы, запросы разработчиков программного обеспечения), но все имеют одинаковую форму. А именно, каждое требование оформляется в виде *блока памяти* со следующими атрибутами:

- размер,
- атрибуты отображения:
  - права доступа,
  - политика кэширования,
- виртуальный адрес (может быть не определён),

- физический адрес (может быть не определён),
- зоны безопасности:
  - передняя зона безопасности (отображаемая или неотображаемая),
  - задняя зона безопасности (отображаемая или неотображаемая),
- флаг непрерывности в физической памяти (должен ли блок отображаться в физическую память непрерывно),
- выравнивание блока в виртуальной памяти.

Блоки могут иметь и другие атрибуты, необходимые для более узких целей, на которых мы сейчас останавливаться не будем.

Заметим, также, что описание блоков хотя бы в некоторой части не зависит от платформы (что позволяет повысить переносимость требований), так что требуются механизмы, позволяющие адаптировать описание требований под конкретную платформу. Такими механизмами являются таблица трансляции прав доступа (отображение общих прав доступа в права доступа реализованные на конкретной платформе) и таблица классов эквивалентности политик кэширования.

Модель программного обеспечения, которую мы используем, связана с моделью ARINC 653: имеется несколько модулей, каждый из которых включает одно ядро и несколько разделов.

Одновременно функционирует ядро и один раздел, при переключении разделов состояние виртуальной памяти меняется в части раздела, а при переключении модулей состояние виртуальной памяти меняется и в части ядра, и в части раздела. То же самое происходит с настройками таблиц отображающих записей (таблиц буфера ассоциативной трансляции (TLB) на платформах типа MIPS [8] или Power PC [9][10] или таблиц страниц на платформах типа RISC-V, семейства Intel x86 [11] или ARMv7 [12]).

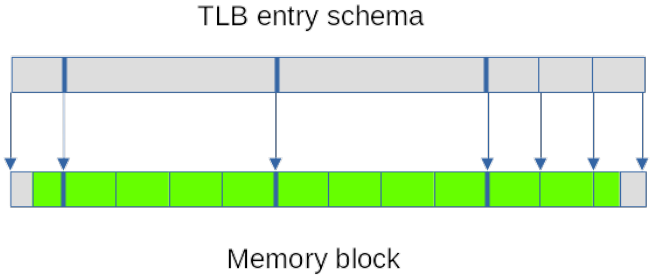


*Рис. 1. Пример схемы отображающих записей, покрывающей блок памяти.*
*Fig. 1. Example of TLB entry schema for memory block.*

Такое строение модели позволяет ввести понятие *частного виртуального пространства адресов*: виртуального пространства, соответствующего конкретному ядру или конкретному разделу и *полного*

*виртуального пространства адресов*, соответствующего сочетанию ядра и раздела.

Для построения статической раскладки памяти необходимо указать место расположения блоков в виртуальной и физической памяти, а также настройки таблиц отображающих записей, обеспечивающих выполнение требований памяти: TLB-таблиц, на платформах с программным управлением TLB или таблиц страниц — на платформах с автоматической настройкой TLB.

Самый общий план построения статической раскладки памяти включает [7]:

- проверку требований на консистентность;
- построение последовательностей (схем) отображающих записей, покрывающих блоки памяти (см. пример на рис. 1);
- размещение построенных записей в виртуальной и физической памяти.

## *3. Экскурс в практику*

В данной статье мы сосредоточимся на аппаратных особенностях, связанных с выделением отдельных областей виртуальной и физической памяти, для которых определяются какие-либо особенности и ограничения на отображения адресов.

Ниже мы представим несколько важных практических сценариев, в которых возникают области памяти с ограничениями на отображения.

### 3.1 Фиксированная трансляция виртуальных адресов

На некоторых платформах, таких как MIPS [8], часть виртуальных адресов транслируется статически с предопределёнными правами доступа. На архитектуре MIPS32 примерами таких отображений являются сегменты kseg0 и kseg1.

Для отображения адресов этих сегментов не нужно использовать отображающие записи из таблицы буфера ассоциативной трансляции (TLB), так как для данных сегментов аппаратно заданы свои собственные отображения. Сегменты kseg0 и kseg1 могут использоваться для доступа к памяти, например, после аппаратного сброса, так как механизм отображения с помощью TLB в этот момент ещё не доступен. Кроме того, размещение блоков памяти в данных сегментах позволяет экономить записи TLB, что существенно для платформ типа MIPS [8], поскольку количество отображающих записей на этих платформах обычно не велико (от 16 до 64).

Отметим важные для нас свойства сегментов kseg0 и kseg1:

- эти диапазоны виртуальных адресов не могут отображаться с помощью записей таблицы TLB;

- отображение адресов данных сегментов фиксировано, последовательно и непрерывно.
- доступ к сегментам kseg0 и kseg1 разрешён только из ядра (привилегированный доступ);
- в каждом сегменте определена своя фиксированная политика кэширования;
- права доступа в данные сегменты определены и фиксированы, имеется доступ ко всем видам операций: чтению, записи и исполнению.

### 3.2 Неатомарное переключение адресных пространств

На платформах, использующих MMU с табличной организацией памяти, легко организовать атомарное переключение адресных пространств, для этого достаточно обновить регистр, содержащий адрес таблицы страниц. Но там, где используется TLB с программным управлением (а это, например, платформы NXP e500mc, NXP e500v2, IBM 470S на архитектуре PowerPC [9] [10]), для переключения адресных пространств приходится программно изменять TLB-записи. А, поскольку код, управляющий MMU, также должен присутствовать в виртуальной памяти, эти программные изменения необходимо производить в строго определённом порядке.

Эту ситуацию можно сильно упростить, выделив для кода, управляющего MMU, несколько фиксированных записей TLB. В этом случае виртуальная память данных записей зарезервирована и не используется для других отображений, и отображение этой памяти в физическую память определено и фиксировано заранее. Таким образом, мы получаем программно заданное отображение. При необходимости, таких отображений можно задать несколько.

В этом сценарии в виртуальной памяти выделяются области со следующими свойствами:

- выделенная область непрерывна в виртуальной памяти;
- отображение адресов выделенной виртуальной области фиксировано, последовательно и непрерывно;
- доступ к выделенной области разрешён только из ядра (привилегированный доступ);
- для выделенной области определена и фиксирована политика кэширования;
- для выделенной области определены и фиксированы права доступа.

### 3.3 Работа с ограничениями IOMMU при DMA-операциях (Direct Memory Access)

Периферийные устройства делают DMA-операции в оперативную память центрального процессора практически на всех известных

архитектурах. Тем не менее, из соображений безопасности или в связи с архитектурными особенностями шины, диапазон доступных периферийному устройству адресов для DMA-операций может не охватывать всю память.

По этой причине в памяти обычно выделяется участок, который считается пригодным для того, чтобы допустить в него периферийные устройства. Это не означает, что всякому периферийному устройству будет дан доступ ко всем адресам данного участка. Вернее будет сказать, что доступ к адресам, не входящим в данный участок памяти точно будет запрещён для периферийных устройств.

Итак, в физической памяти появляются области со следующими свойствами:

- область непрерывна в физической памяти, имеет фиксированный начальный адрес и размер;
- некоторые области виртуальных адресов (отвечающих периферийным устройствам) должны быть отображены только в эту область.

### 3.4 Неконфигурируемые ядром политики кэширования

Для работы DMA в непривилегированном режиме необходимо обеспечивать когерентность памяти. Часто это приходится делать вручную посредством очистки кэшей процессора. Хотя архитектура RISC-V и микроядерные ОС массово используются в современной промышленности, далеко не все современные реализации процессоров RISC-V предоставляют возможность управлять содержимым кэшей процессора из непривилегированного режима (см. расширение Zicbom).

На таких платформах ускорить код DMA-операций помогает задание политики кэширования на уровне области памяти. Так как в стандартном исполнении 32-битной архитектуры RISC-V в S-режиме, в котором традиционно исполняется ядро операционной системы, отсутствует возможность задания политики кэширования при настройке MMU, такие области памяти удобно конфигурировать до старта операционной системы. Для этого вся память платформы разделяется на сегменты, в которых политика кэширования фиксируется.

Выделенные здесь области физической памяти обладают следующими свойствами:

- область непрерывна в физической памяти, имеет фиксированный начальный адрес и размер;
- разрешается отображать виртуальные адреса в адреса данной области только с определённой политикой кэширования.

### 3.5 Арифметическая адресация устройств для шины PCI Express

Контроллеры PCI Express, поддерживающие ECAM-адресацию устройств, для отображения конфигурационного пространства в физической

памяти используют простую арифметическую зависимость, в которой адрес устройства или функции линейно зависит от их индексов (см. пример на рис. 2 ). Такой способ адресации существенно упрощает работу с иерархией устройств в операционной системе. При использовании виртуальной памяти в ядре обычно сохраняют данное ценное свойство.

При переносе драйверов PCI Express устройств в пользовательское адресное пространство часто возникает необходимость разграничивать права доступа, в том числе в части конфигурационного пространства. Тем не менее, если количество драйверов больше одного, общая схема адресации может быть полезна и здесь.

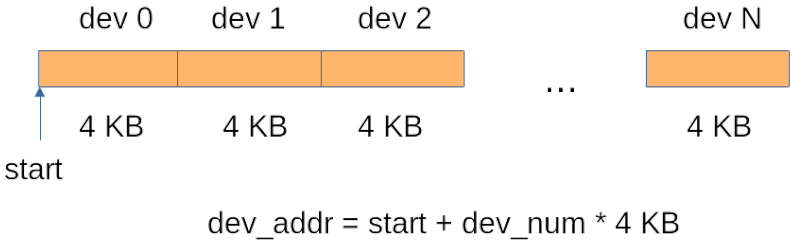


*Рис. 2. Арифметическая адресация устройств.*
*Fig. 2. Address arithmetic for devices.*

Для физической области, содержащей информацию об устройствах, должны выполняться свойства:

- область непрерывна в физической памяти, имеет фиксированный начальный адрес и размер;
- виртуальные области адресов, отвечающих периферийным устройствам, должны быть отображены в эту область;
- в виртуальной памяти должна сохраняться арифметическая адресация устройств.

Заметим, что в этом случае физические адреса устройств заранее известны.

### 3.6 Работа с адресами при DMA-операциях

Арифметическая адресация может быть также полезна при DMA-операциях. При работе с входящими и исходящими пакетами данных в сетевом стеке или с блоками данных в подсистеме криптоускорителей часто возникает потребность в эффективном получении адреса на шине периферийного устройства.

Даже простое ручное преобразование одного виртуального адреса в физический может вызвать заметные просадки производительности, а когда речь идёт о больших объёмах данных, распределённых по нескольким

страницам, переменное количество дескрипторов затрудняет расчёт необходимой оперативной памяти под дескрипторы, а также делает вычисление WCET на их формирование достаточно трудным.

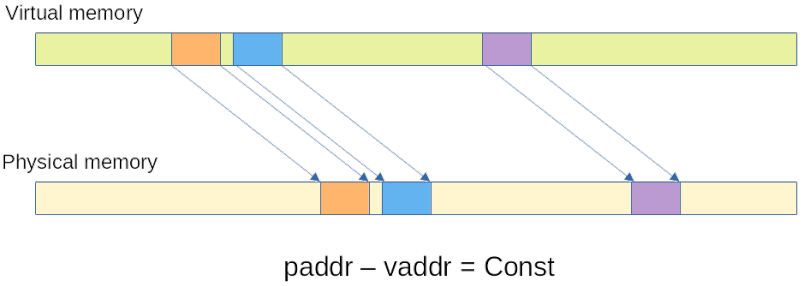


*Рис. 3. Отображение, сохраняющее сдвиг адресов (изометрическое отображение).*
*Fig. 3. Mapping with constant address shift (isometric mapping).*

Понятной альтернативой является линейное отображение виртуальной памяти на физическую в заданном диапазоне, что, как правило, позволяет получить адрес на шине одной арифметической операцией. Свойство отображения сохранять сдвиг адресов называется *изометричностью*, так как при таком отображении расстояния между отображаемыми адресами не меняются (см. рис. 3).

Области физической памяти, которые возникают в этом случае, имеют свойства:

- область непрерывна в физической памяти;
- отображение из виртуальной памяти в физическую может быть частичным для данной области, но сохраняет порядок адресов в областях, соответствующих сетевым буферам;
- допускается использование различных прав доступа в областях для буферов различного назначения.

## *4. Построение общей модели для областей с ограничениями на отображения*

В примерах, приведённых в предыдущем разделе, уже выделено главное понятие, которое возникает во всех указанных случаях. Это *область с ограничениями на отображения*.

Такие области могут выделяться в виртуальной или физической области. Поэтому, имеет смысл разделить их на две категории:

- виртуальные области с ограничениями на отображения (ВООО),
- физические области с ограничениями на отображения (ФООО).

Это разделение тем более имеет смысл, что накладываемые ограничения для случаев виртуальной и физической области различаются. А именно, для виртуальных областей наложение ограничений означает задание конкретного отображения на этой области, а для физических областей

ограничения накладываются на атрибуты отображения, но при этом сами отображения явно не задаются.

### 4.1 Виртуальные области с ограничениями на отображения

Как было упомянуто ранее, для BOOO отображение задаётся явно. Это отображение будем называть *фиксированным отображением данной BOOO*. Фиксированное отображение отображает виртуальную область в физическую память непрерывно и последовательно (см. рис. 4). Для блоков памяти, попавших в виртуальную область с ограничениями на отображения не требуется задавать дополнительные отображения, помимо фиксированного отображения данной BOOO.

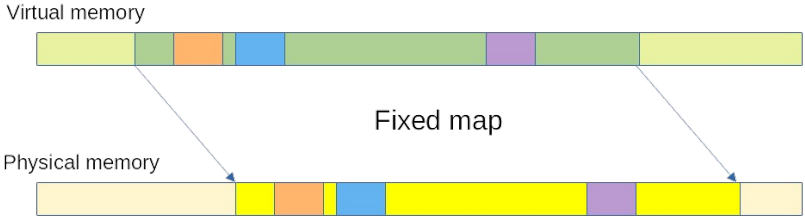


*Рис. 4. Фиксированное отображение BOOO.*
*Fig. 4. Fixed mapping for virtual area with mapping constraints.*

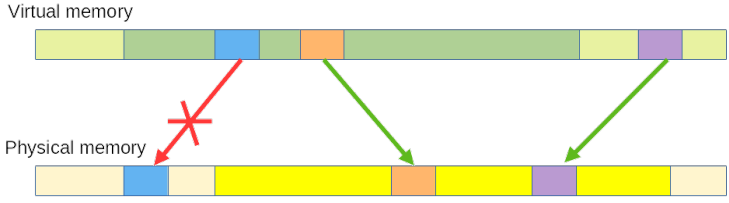


*Рис. 5. Допустимые и недопустимые для BOOO отображения блоков.*
*Fig. 5. Allowed and unresolved block mappings for a virtual area with mapping constraints.*

Таким образом, для виртуальных областей с ограничениями на отображения ключевыми атрибутами являются:

- размер области,
- начальный виртуальный адрес области,
- начальный физический адрес образа области в физической памяти,
- ограничения на атрибуты отображения:
  - разрешённые права доступа,
  - разрешённые политики кэширования.

Поскольку при отображении виртуальных адресов не допускается неоднозначность атрибутов отображения, то политики кэширования, указанные в списке разрешённых политик должны быть эквивалентны, а варианты прав доступа, указанные в списке разрешённых прав доступа должны транслироваться в одни и те же права на данной платформе.

Что касается блоков, которые попали в физический образ целевой ВООО, то для них нет ограничений на атрибуты отображения, если в виртуальной памяти они не находятся в памяти целевого ВООО (см. рис 5).

Фиксированное отображение обладает свойством изометричности.

Для того, чтобы указать, что определённый блок памяти обязан находиться в виртуальной памяти некоторого ВООО, у такого блока памяти будем задавать специальный атрибут — родительскую область.

Примеры практических сценариев, в которых возникают виртуальные областей с ограничениями на отображения, приведены в разделах 3.1 и 3.2.

На рисунке 6 приведёно описание ВООО для сегмента kseg0.

```
type: virtual # виртуальная область памяти
name: kseg0 # область сегмента kseg0
size: 0x20000000 # размер в байтах
vaddr: 0x80000000 # виртуальный адрес сегмента
paddr: 0 # начальный адрес образа сегмента в физической памяти
constraints: # ограничения на отображения
  kernel: # отображение разрешено только для ядра
    allowed_cache_policy: # допустимые политики кэширования
    - DEFAULT # политика по умолчанию
    allowed_permissions: # допустимые права доступа
    - RWX # права доступа на чтение, запись, исполнение
```

*Рис. 6. Пример описания ВООО для сегмента kseg0.*
*Fig. 6. Example of virtual area description for kseg0.*

Физические образы разных ВООО могут пересекаться или даже совпадать. Но сами виртуальные области на большинстве платформ пересекаться не должны.

Платформенно независимое описание означает, что виртуальная область может не лежать целиком в виртуальной памяти. В этом случае, блоки данного ВООО должны располагаться в пересечении виртуальной памяти и памяти целевой ВООО.

### 4.2 Физические области с ограничениями на отображения

Теперь посмотрим, какие атрибуты нужны для физических областей с ограничениями на отображения.

Здесь, как говорилось выше, отображение не задаётся явно, таким образом, виртуальные адреса могут отображаться в данную область через несколько отображений, которые обычно задаются отображающими записями. Ограничения, которые задаются, будут определяться списком

разрешённых политик кэширования (это позволит, например, описать области из примера 3.3) и списком разрешённых прав доступа.

Напомним, что для некоторых физических областей с ограничениями на отображения имеется условие сохранения арифметической адресации (примеры 3.5, 3.6). Рассмотрим это требование подробнее.

Во-первых, арифметическая адресация касается не всех блоков, которые могут попасть в определяемую физическую область, а только некоторых (для примера 3.5 это будут блоки памяти для периферийных устройств). Таким образом, нам нужен механизм, сообщающий, что некоторые блоки памяти относятся к данной ФООО, а другие не относятся. Эту связь можно установить так же, как мы это сделали для виртуальных областей, определив у блока специальный атрибут — родительскую область с ограничениями на отображения.

Во-вторых, сохранение арифметической адресации означает сохранение одинакового порядка адресов при трансляции из виртуальной памяти в физическую. Иначе говоря, трансляция адресов для блоков памяти, относящихся к данному ФООО, должна сохранять расстояния между транслируемыми адресами, то есть должна быть изометрическим отображением.

Следует заметить, что требование изометричности отображения шире, чем просто арифметическая адресация для блоков.

```
type: physical # физическая область памяти
name: pci0_cfg # область контроллера PCI0
size: 0x100000 # размер области
paddr: 0xf4000000 # физический адрес области
constraints: # ограничения на отображения
  kernel: # ограничения для отображений из привилегированной памяти
    allowed_cache_policy: # допустимые политики кэширования
    - IO # политика кэширования, совместимая с MMIO
    allowed_permissions: # допустимые права доступа
    - R # права доступа на чтение
    - W # права доступа на запись
    - RW # права доступа на чтение и запись
  partition: # ограничения для отображений из непривилегированной памяти
    allowed_cache_policy: # допустимые политики кэширования
    - IO # политика кэширования, совместимая с MMIO
    allowed_permissions: # допустимые права доступа
    - R # права доступа на чтение
    - W # права доступа на запись
    - RW # права доступа на чтение и запись
  isometric_address_translation: true # изометрическая трансляция адресов
```

*Рис. 7. Пример описания ФООО для области контроллера PCI0.*
*Fig. 7. Example of physical area description for PCI0 controller.*

Таким образом, описание физической области с ограничениями на отображения должно включать следующие атрибуты:

- размер отображения,
- начальный физический адрес отображения,
- ограничения на отображения:

- на атрибуты отображения для ядра:
  - разрешённые права доступа
  - разрешённые политики кэширования
- на атрибуты отображения для раздела:
  - разрешённые права доступа
  - разрешённые политики кэширования
- атрибут, указывающий должно ли выполняться требование изометричности для отображений, задаваемых блоками данной ФООО.

Заметим, что ограничения могут быть вообще не указаны, в этом случае семантика такой области будет состоять в требовании, что блоки, относящиеся к данной ФООО должны находиться в ней, как это описано в примере 3.4.

На рисунке 7 дан пример описания физической области с ограничениями на отображения для области контроллера PCI0.

## *5. Построение раскладки памяти для блоков, относящихся к областям с ограничениями на отображения*

Для построения раскладки памяти в областях с ограничениями на отображения выделим три случая:

- виртуальные области с ограничениями на отображения,
- физические области с ограничениями на отображения без требования изометричности отображений,
- физические области с ограничениями на отображения с требованием изометричности отображений.

Разберём эти случаи по отдельности.

### 5.1 Виртуальные области с ограничениями на отображения

Поскольку ВООО своим фиксированным отображением отображается в физическую память непрерывно и последовательно, то расположение блоков внутри самой ВООО и её образа в физической памяти совершенно одинаково. Таким образом, мы можем не различать эти области (ВООО и её физический образ) в процессе размещения блоков внутри них.

Далее, есть блоки, у которых заранее известен виртуальный адрес, или физический адрес, или оба адреса вместе. Если при этом блок попадает в нашу ВООО, то, поскольку других отображений из этой области нет, то блок этот отображается известным образом в физическую память и физический адрес его известен. Таким образом, нам известно, какой именно участок памяти уже занят и мы можем его исключить из рассмотрения.

Если же у блока A известен физический адрес и блок частично или полностью попадает в физический образ нашего ВООО, то в тех виртуальных адресах, которые отображены в физическую память блока A

нельзя размещать никакие другие блоки, так как иначе возникнет пересечение с физической памятью блока A. Таким образом, эту память также можно исключить из рассмотрения.

Таким образом, доступная для размещения блоков память виртуальной области выглядит как система свободных интервалов и задача размещения получает следующую формулировку.

*Пусть заданы интервалы памяти. И пусть заданы блоки памяти, для которых указан их размер и выравнивание. Нашей целью является размещение блоков в указанных интервалах с соблюдением выравнивания.*

В этой формулировке скрыта некоторая неясность. А именно, выравнивание зависит от конкретных адресов интервалов, а мы рассматриваем не виртуальную или физическую память, а некую их "склейку". Эта проблема легко решается, если в требованиях задано выравнивание блока в виртуальной памяти. В этом случае мы просто размещаем блоки так, как если бы они размещались в виртуальной памяти, а физические адреса вычисляем согласно фиксированному отображению нашей BOOO. Если же добавляется выравнивание блоков в физической памяти, то здесь необходим строгий контроль соответствия виртуальных и физических адресов, так как сдвиг адресов, определяемый фиксированным отображением не должен конфликтовать с выравниванием блока в виртуальном и физическом пространстве.

Приведём пример. Пусть выравнивание адреса блока в виртуальной памяти идёт на 4 бита, а выравнивание в физической памяти — на 6 бит (выравнивание блоков обычно указывается в байтах, но здесь для упрощения примера мы вводим выравнивание по битам, для уменьшения количества рассматриваемых битов):

- если при этом сдвиг физического адреса относительно виртуального составляет $64 = 1000000_2$, то всё в порядке, только следует потребовать выравнивание в виртуальной памяти на 6 битов;
- если при этом сдвиг физического адреса относительно виртуального составляет $32 = 100000_2$, то требование усложняется, нужно, чтобы адрес блока в виртуальной памяти имел последние биты, равные '110000', тогда добавив сдвиг 32 мы всегда получим выровненный на 6 битов адрес в физической памяти;
- если же при этом сдвиг физического адреса относительно виртуального составляет $12 = 1100_2$, то выровнять виртуальный адрес на 4 бита, и физический адрес на 6 бит никак не удастся, возникает неразрешимый конфликт.

Таким образом, нам следует ввести понятие *выравнивания блока в общем смысле*, которое будет означать не только обнуление нескольких битов начального адреса блока, но требование о фиксированной последовательности нескольких последних битов этого адреса, не

обязательно нулевой. Другими словами, для блока нужно указать *хвост адреса*, то есть конкретные значения определённого числа последних битов.

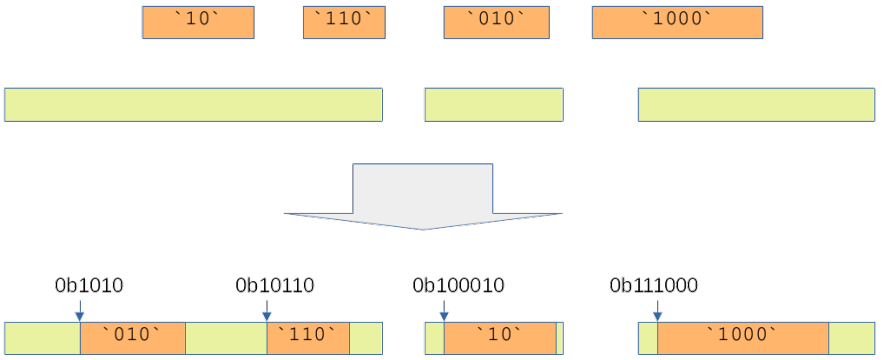


*Рис. 8. Задача размещения блоков с выравниванием адресов.*
*Fig. 8. Memory allocation problem for blocks with alignment.*

Итак, уточним формулировку задачи размещения (см. рис. 8).

**Задача размещения с учётом выравнивания**. *Пусть заданы интервалы памяти. И пусть заданы блоки памяти, для которых указан их размер и выравнивание в общем смысле (хвост адреса). Нашей целью является размещение блоков в указанных интервалах так, чтобы блоки не пересекались и виртуальные адреса блоков имели бы соответствующие хвосты.*

Данную задачу можно связать с классическими задачами о рюкзаке или построении максимальной клики в графе, так что построить эффективное точное решение здесь не удастся. Однако, для практических целей можно применять некоторые приёмы, позволяющие находить решение, если и не во всех случаях, то во многих, возникающих в реальных задачах.

Обсуждению этой темы мы планируем посвятить отдельную работу, в настоящий момент готовящуюся к печати.

### 5.2 Физические области с ограничениями на отображения без требования изометричности отображений

Если не требуется сохранять расстояния между адресами в виртуальной области (см. рис. 9), то размещение блоков в памяти физических областей с ограничениями на отображения может осуществляться на этапе размещения соответствующих покрывающих записей. В этом случае необходимо добиться того, чтобы все записи, покрывающие блок данной ФООО оказались в физической памяти этой ФООО.

Здесь задача будет проще, так как записи всегда являются двоичными страницами — это означает, что адрес записи всегда пропорционален её размеру, который является степенью двойки. Легко показать, что любые две

двоичные страницы либо содержат одна другую целиком, либо никак не пересекаются. Такая иерархичность двоичных страниц, а, значит, и отображающих записей, позволяет упрощать их размещение в физической памяти. Для этого, например, можно устраивать размещение в порядке убывания размеров записей.

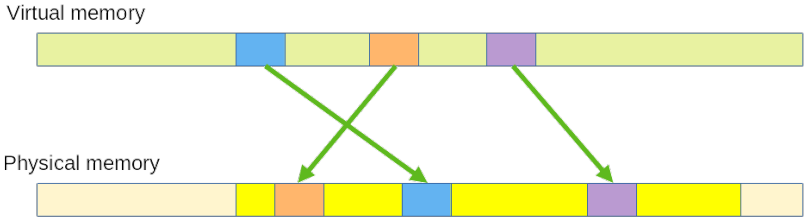


*Рис. 9. Физическая область с ограничениями на отображения без требования изометричности отображения адресов.*
*Fig. 9. Physical area with mapping constraints without isometric address translation.*

Этот метод хорош, когда можно размещать записи по отдельности. Если же записи нельзя отрывать друг от друга (например, если покрываемый ими блок обязан быть непрерывным в памяти), то должны быть использованы те же методы, что и в предыдущем пункте.

### 5.3 Физические области с ограничениями на отображения с требованием изометричности отображений

Посмотрим на ФООО с требованием изометричности с точки зрения частного виртуального пространства, то есть виртуального адресного пространства конкретного ядра или конкретного раздела.

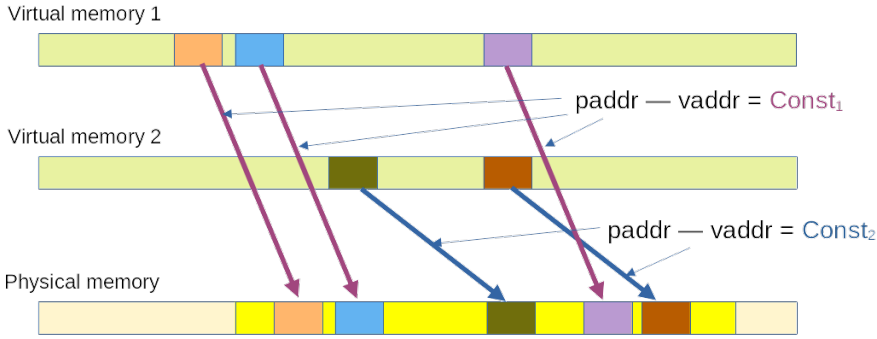


*Рис. 10. Физическая область с ограничениями на отображения с требованием изометричности отображения адресов.*
*Fig. 10. Physical area with mapping constraints with isometric address translation.*

В каждом таком виртуальном пространстве могут быть блоки, принадлежащие данной физической области с ограничениями на отображения (см. рис. 10).

Поскольку требуется, чтобы в этом виртуальном пространстве выполнялось требование изометричности отображений для блоков данного ФООО, то, если есть хотя бы один блок, для которого известен как виртуальный, так и физический адрес, эти два адреса будут задавать сдвиг адресов для всех блоков данного ФООО этого частного виртуального пространства.

Например, пусть есть виртуальное пространство (m1, P1) модуля m1, раздела P1. И пусть в этом виртуальном пространстве есть блоки A и B, которые принадлежат заданному ФООО. Если у блока A известны виртуальный адрес 5 и физический адрес 37, а у блока B известен виртуальный адрес 12, то для выполнения свойства изометричности физический адрес блока B обязан равняться 44 (см. рис. 11).

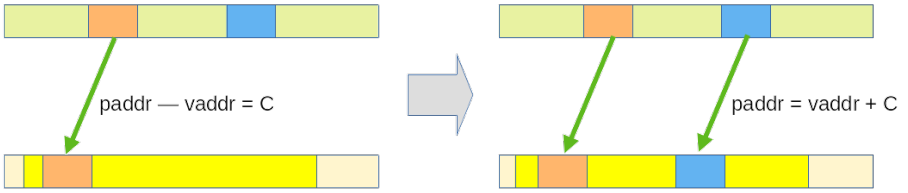


*Рис. 11. Вычисление адресов блоков при наличии неявных зависимостей.*
*Fig. 11. Calculating block addresses in the presence of implicit dependencies.*

Таким образом, можно вычислить некоторые адреса блоков, исходя из адресов других блоков и свойства изометричности, и свести ситуацию в данном частном виртуальном пространстве к одному из следующих случаев:

а) в виртуальном пространстве нет блоков данного ФООО;

б) в виртуальном пространстве ровно один блок данного ФООО;

в) в виртуальном пространстве для части блоков данного ФООО известны оба адреса — виртуальный и физический, а для остальных блоков данного ФООО оба адреса неизвестны;

г) в виртуальном пространстве для части блоков данного ФООО известен только виртуальный адрес, для части блоков известен только физический адрес, а для остальных оба адреса неизвестны.

Случай а) не представляет интереса. Случай б) сводится к предыдущему пункту (когда изометричность не требуется), так как для одного блока эти ситуации неразличимы.

Для того, чтобы разместить прочие блоки предлагается вычислить их виртуальные и физические адреса до того, как будет построена модель покрывающих блоки записей.

Блок с известным виртуальным адресом будем называть *фиксированным в виртуальном пространстве*. Блок с известным физическим адресом будем называть *фиксированным в физическом пространстве*. Блок с известными виртуальным и физическим адресом будем называть *полностью фиксированным*. Блоки с неизвестными виртуальным и физическим адресами будем называть *нефиксированными*.

**В виртуальных пространствах типа в)** полностью фиксированные блоки уже размещены в виртуальном и физическом пространстве. Для таких виртуальных пространств известен сдвиг адресов, если есть хотя бы один полностью фиксированный блок.

**В виртуальных пространствах типа г)** блоки, фиксированные только в виртуальном пространстве, образуют цепочку, размещать которую в физическом пространстве можно только целиком, так как требуется сохранить расстояния между блоками. То же самое касается блоков фиксированных только в физическом пространстве.

Итак, в каждом виртуальном пространстве остались следующие не размещённые блоки:

- цепочка блоков, фиксированная в виртуальном пространстве;
- цепочка блоков, фиксированная в физическом пространстве;
- нефиксированные блоки.

Каждая из этих категорий блоков может быть пустой.

Далее необходимо провести следующие операции:

- в пространствах, где есть фиксированные цепочки в виртуальной памяти и в физической памяти, разместить обе эти цепочки таким образом, чтобы
  - в физической памяти не было пересечений с занятыми участками (памятью блоков),
  - в виртуальной памяти не было пересечений с занятыми участками (памятью блоков и зонами безопасности);
- в пространствах, где есть только фиксированная цепочка в виртуальной памяти, разместить эту цепочку в физической памяти так, чтобы не было пересечений с занятыми участками;
- в пространствах, где есть только фиксированная цепочка в физической памяти, разместить эту цепочку в виртуальной памяти так, чтобы не было пересечений с занятыми участками;

После проведения этих этапов, в каждом частном виртуальном пространстве могут быть только блоки следующих категорий:

- полностью фиксированные блоки;
- нефиксированные блоки.

Если есть полностью фиксированные блоки, то сдвиг адресов известен, так что размещение нефиксированных блоков можно производить так же как

это делалось для виртуальных областей в пункте 5.1. То есть создать «слитую» воедино модель адресного пространства и размещать нефиксированные блоки с учётом их выравнивания в этом объединённом пространстве, сразу получая для блоков и виртуальные и физические адреса.

Если же полностью фиксированных блоков нет, то размещение нефиксированных блоков можно производить сначала только в физическом пространстве (так как оно обычно более дефицитно), а потом, полученную фиксированную в физической памяти цепочку размещать в виртуальной памяти как единый объект.

В данной схеме решения как отдельный этап можно выделить следующую задачу.

Пусть имеется базовый интервал памяти, в котором указаны свободные и занятые места (подинтервалы). И пусть имеется цепочка блоков, которая должна быть размещена в свободных местах данного интервала. Эта цепочка может быть размечена таким же образом — как интервал со свободными и занятыми участками. Адрес цепочки-интервала должен удовлетворять требованию выравнивания общего вида.

Задача состоит в том, чтобы найти все точки базового интервала в которых можно разместить цепочку, такие, что занятые участки цепочки не пересекаются с занятыми участками базового интервала и начальный адрес цепочки соответствует её выравниванию (то есть точки с хвостом адреса таким же, как у цепочки).

Схематично решение можно описать так.

Проводится параллельный проход по точкам разметки интервала и точкам разметки цепочки.

Сначала рассматривается первая свободная точка, подходящая для начала первого участка цепочки. В критерий подходящей входит соответствие хвостов адресов, то есть выравнивания цепочки в общем смысле (см. предыдущие пункты). Для этой точки проводится проверка по точкам разметки цепочки, при проверке сохраняется информация о максимальном возможном свободном сдвиге.

Если обнаружен конфликт (пересечение занятых участков), то вычисляется сдвиг цепочки, пропускается участок, на котором пересечение обязательно будет, и происходит переход к следующей точке-кандидату для вставки.

Если же конфликтов не обнаружено, то по собранной информации о возможном свободном сдвиге вычисляется допустимый интервал точек-вставок и происходит переход к следующей точке-кандидату.

Таким образом, допустимые точки-вставки хранятся компактно в виде интервалов, а не отдельных точек.

Сложность данного алгоритма составляет O(mn), где m — количестов точек разметки интервала памяти, а n — количество точек разметки фиксированной цепочки.

Поскольку размер одной статьи ограничен, более подробным описаниям алгоритмов для случая ФООО с требованием изометрической трансляции адресов мы намерены посвятить отдельную работу.

## *6. Заключение*

В работе проводится анализ аппаратных особенностей различных платформ, которые могут быть описаны в общих терминах областей памяти с ограничениями на отображения, строится общая модель и обсуждаются подходы к построению статической раскладки памяти для таких случаев.

Предложенная методика позволяет расширить концепцию статического распределения памяти для различных сценариев работы с устройствами, такими как MMIO или DMA, а также справляться с нетривиальными архитектурными особенностями работы MMU на разных платформах. Одним из преимуществ, которые реализуются в рамках данной методики, является возможность строить на статике отображения определённых блоков памяти, сохраняющие сдвиг адресов. Такая организация отображений сводит трансляцию адресов к линейному преобразованию, что позволяет упрощать архитектуру драйверов и добиваться лучшей предсказуемости по времени.

Некоторые аспекты задачи статической раскладки памяти для описываемых структур могут быть сформулированы в строгих терминах, как расширение классических задач комбинаторной оптимизации. Хотя эффективного точного решения таких задач не существует, для них можно строить эвристические методы решения, которые хорошо подходят для практического применения.

Метод был успешно применён в семействе ОСРВ КЛОС, которые используются в различных сегментах аэрокосмической отрасли.

## *7. Список литературы*

[1]. ARINC Industry Activities. ARINC Specification 653P0-3. Avionics Application Software Standard Interface Part 0. Overview of ARINC 653. AEEC. 2019-08-07. 45 p.

[2]. ARINC Industry Activities. ARINC Specification 653P1-5. Avionics Application Software Standard Interface Part 1. Required Services. AEEC. 2019-12-23. 295 p.

[3]. Khoroshilov A.V., Cheptsov V.Y. Robust Resource Partitioning Approach for ARINC 653 RTOS.

[4]. Бурдонов И.Б., Косачев А.С., Петренко А.К., Хорошилов А.В., Чепцов В.Ю. Семейство операционных систем КЛОС. *Труды*

*Института системного программирования РАН*. 2025;37(6):11-26. https://doi.org/10.15514/ISPRAS-2025-37(6)-47

[5]. И.Б. Бурдонов, А.С. Косачев, В.Н. Пономаренко. Операционные системы реального времени. Препринт ИСП РАН 14, 2006 г. http://burdonov.ru/doctor/papers_2006/Operatsionnye_sistemy_realnogo_vremeni/Operatsionnye_sistemy_realnogo_vremeni.pdf

[6]. Mallachiev K.M., Pakulin N.V., Khoroshilov A.V. Design and architecture of real-time operating system. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 2, 2016, pp. 181-192. DOI: 10.15514/ISPRAS-2016-28(2)-12.

[7]. С.А. Зеленова. Статическое распределение памяти для операционных систем реального времени. Труды ИСП РАН, том 36, вып. 3, 2024 г.

[8]. Heinrich J. MIPS R4000 Microprocessor User's Manual. Prentice-Hall, June 1993. http://groups.csail.mit.edu/cag/raw/documents/R4400_Uman_book_Ed2.pdf

[9]. [PPC500] e500mcRM revision 3. e500mc Core Reference Manual. Freescale Semiconductor, Inc. 2013-03. 440 p. https://www.nxp.com/docs/en/reference-manual/E500MCRM.pdf

[10]. [PPC400] PowerPC 476FP Embedded Processor Core User’s Manual. International Business Machines Corporation. 2014-01-10. 320 p.

[11]. Intel 64 and IA-32 Architectures Software Developer’s Manual Combined Volumes 3A, 3B, 3C, and 3D: System Programming Guide. 2023-09. 1536 p. https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html

[12]. [ARM] Cortex-M4 Revision r0p0. Technical Reference Manual. ARM Limited. 2010-03-02. 117 p.