DOI: 10.15514/ISPRAS-2024-36(3)-3

Статическое распределение памяти для операционных систем реального времени

С.А. Зеленова, ORCID: 0000-0002-0776-9651 <*sophia@ispras.ru*>

Институт системного программирования РАН,
Россия, 109004, г. Москва, ул. А. Солженицына, д. 25.

Аннотация. Задача повышения надежности критических операционных систем реального времени (ОСРВ) не теряет актуальности. Использование детализации требований разработчиков дает новые возможности в этом направлении при управлении памятью [9]. В статье представлен новый подход к статическому распределению памяти в ОСРВ с надёжной изоляцией по памяти. Предлагается метод построения инструмента статической раскладки памяти по формальному описанию требований разработчика на память. Формальное описание предлагается строить без привязки к платформе, а основываясь только на нуждах разрабатываемого ПО. Вводятся общие понятия, дающие возможность универсального подхода к построению инструмента статической раскладки, описана общая схема алгоритма раскладки, детализируются требования, которые необходимо учитывать при реализации каждого шага алгоритма. Описываемый подход апробирован на реальных индустриальных проектах и показал свою универсальность, адаптивность и эффективность в построении статической раскладки памяти.

Ключевые слова: операционные системы реального времени; статическая раскладка памяти; автоматизация раскладки памяти; формальные требования на память.

Для цитирования: Зеленова С.А. Статическое распределение памяти для операционных систем реального времени. Труды ИСП РАН, том 36, вып. 3, 2024 г., стр. 35–48. DOI: 10.15514/ISPRAS–2024–36(3)–3.

Static Memory Allocation for Real-Time Operating Systems

S.A. Zelenova, ORCID: 0000-0002-0776-9651 <*sophia@ispras.ru*>

Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.

Abstract. Critical real-time operating system (RTOS) reliability improvement remains to be a relevant and demanding task. The use of detailed requirements provided by the developers introduces new opportunities in this direction through the memory management facilities. In this paper we present a new approach of static memory allocation in real-time systems with robust memory space partitioning. We propose to design a static memory layout tool based on the formal description of the project memory requirements. The proposed formal requirements are platform agnostic and are based only on the needs of the application software. We introduce general concepts, which allow us to use a universal approach for static memory layout tool creation. We also describe the general scheme of the memory layout algorithm as well as the requirements that must be taken into account when each step of the algorithm is implemented. We tested our approach on real industrial projects and confirmed its versatility, adaptability and effectiveness.

Keywords: real-time operating system; static memory allocation; memory allocation methods; robust partitioning.

For citation: Zelenova S.A. Static memory allocation for real-time systems. *Trudy ISP RAN/Proc. ISP RAS*, vol. 36, issue 3, 2024. pp. 35-48 (in Russian). DOI: 10.15514/ISPRAS-2024-36(3)-3.

1. Введение

Среди систем реального времени выделяется широкий ряд систем, цена ошибок для которых весьма высока. Классический пример – авионика. Для таких систем критическими факторами являются и время отклика, и надёжность. Причем для временного фактора на первый план выходит предсказуемость, а не скорость.

В авионике в результате перехода от федеративной бортовой сети, состоящей из множества вычислителей, выполняющих одну задачу, к бортовой сети на основе интегрированной модульной авионики (ИМА), удалось значительно снизить стоимостные и весовые характеристики бортового оборудования. Тем не менее, так как в концепции ИМА на каждом вычислителе выполняется множество задач разной степени критичности, для обеспечения надёжной изоляции процессов по стандарту ARINC 653 в используемых в авионике операционных системах реального времени применяют не только временную, но и пространственную изоляцию.

Для того, чтобы обеспечить надёжную изоляцию процессов с помощью операционной системы реального времени, необходимо заранее построить конфигурацию памяти, поскольку критической ОСРВ нужны твёрдые гарантии доступности необходимых ресурсов в любой момент её работы [3].

Таким образом, естественным решением является статическое распределение памяти в рамках подготовки системы к работе. Часто алгоритмы статического распределения памяти являются тривиальными, что приводит к перерасходу ресурсов и их нехватке. В качестве решения проблемы обычно предлагаются два пути:

- пересмотр требований разработчиков в сторону уменьшения используемых ресурсов, что не всегда возможно;
- увеличение аппаратных возможностей, что ведет к удорожанию проектов, а иногда и невозможно, в силу физических причин (превышение допустимого объёма/массы разрабатываемого оборудования, перегрев оборудования или превышение допустимых пределов времени доступа к памяти в силу конструктивных особенностей оборудования).

Однако на этапе статического распределения можно достичь некоторых важных целей [9]:

- снизить накладные расходы на поиск свободной памяти путём выделения памяти под конкретные задачи и, таким образом, сделать время работы системы более предсказуемым;
- повысить безопасность и надежность системы за счёт целенаправленной защиты критичных участков памяти;
- расширить возможности разработки без привлечения дополнительных ресурсов за счёт более экономного использования памяти.

Согласно подходу, предложенному в [9], для достижения указанных целей статическая конфигурация памяти должна включать в себя сведения не только о том, где находится сам раздел, но и определять способ обращения к участкам памяти, соответствующим частям этого раздела. Таким образом, задача построения становится гораздо сложнее примитивного разделения памяти на непересекающиеся куски, но достоинства данного подхода перевешивают сложности.

Целью данной работы является

- ввести систему понятий, которая может использоваться для построения статической раскладки памяти, нацеленной на повышение надежности системы;
- показать, какие трудности могут встретиться на пути построения статической раскладки памяти;
- дать основу для решения данной задачи;

- продемонстрировать работоспособность предлагаемого подхода.

Работа состоит из введения, трёх разделов и заключения. В первом разделе дан обзор особенностей аппаратной поддержки управления памятью на современных платформах, во втором разделе описана общая схема работы алгоритма статического распределения памяти, в третьем приведены практические результаты применения предлагаемого подхода.

2. Особенности управления памятью

Обсудим некоторые способы организации управления памятью, используемые в популярных архитектурах. В данной работе мы обсуждаем статическое распределение памяти для платформ, использующих для ограничения несанкционированного доступа механизм виртуальной памяти. Однако, как показывает наша практика, похожие алгоритмы статической раскладки применимы и для других платформ, на которых понятие виртуальной памяти отсутствует.

2.1 Двоичные страницы

Практически все современные архитектуры пользуются следуюшим разделением линейных участков памяти на интервалы, которые мы будем называть двоичными страницами.

Именно, *двоичная страница* — это непрерывный интервал памяти какого-либо типа, имеющий размер равный степени двойки и начальный адрес кратный размеру. Так, двоичная страница размера 4 может начинаться в адресах 0, 4, 8, 16 и т.д., но не может начинаться в адресах 2 или 11, так как они не кратны 4 (см. рис. 1).

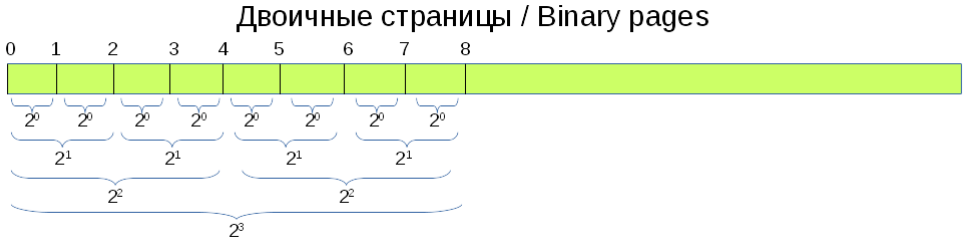


Рис. 1. Двоичные страницы
Fig. 1. Binary pages

Свойство иерархичности. Из определения следует, что, если взять две двоичные страницы в каком-либо отрезке какого-либо вида памяти, то они либо никак не пересекаются, либо меньшая содержится в большей целиком. Равные по размеру двоичные страницы либо совпадают, либо не пересекаются. На рис. 1 показана иерархия двоичных страниц.

В классической литературе [1] двоичные страницы в виртуальной памяти называются просто страницами, а двоичные страницы в физической памяти называются страничными блоками или фреймами (page frame). Здесь мы будем также использовать словосочетания «виртуальная страница» и «физическая страница» как синонимы этих понятий.

Отображение виртуальной памяти в физическую обычно организуется с помощью отображающих записей. Такая запись отображает двоичную страницу в виртуальной памяти на двоичную страницу такого же размера в физической памяти непрерывно и последовательно, так что отображение адресов страницы может быть вычислено по формуле:

целевой физический адрес = начальный адрес физической страницы +
(транслируемый виртуальный адрес – начальный адрес виртуальной страницы)

Для отображающей записи указываются свойства отображения — политика кэширования и права доступа. Отображающие записи организуются в таблицы двумя способами: в таблицы страниц и/или таблицы TLB.

2.2 Таблицы TLB (translation lookaside buffer)

Во многих современных архитектурах для перевода виртуальных адресов в физические используются таблицы записей TLB. TLB является ассоциативной памятью [2] и, таким образом, его использование существенно ускоряет поиск нужного физического адреса.

Таблицы TLB содержат фиксированный набор записей, отображающих виртуальные адреса на физические. Атрибуты отображающих записей такие, как размер, политика кэширования, права доступа, виртуальные и физические адреса, могут быть настраиваемыми или фиксированными, в зависимости от конкретной архитектуры. Таблицы TLB обычно содержат до 4096 записей, для встроенных систем количество TLB-записей может быть весьма незначительным (например, 16, 32 или 64 записи).

TLB может заполняться как программно, посредством явного заполнения записей с установленными атрибутами программным кодом, так и аппаратно, автоматически при обращении к адресам в режиме вытеснения давно неиспользуемых записей (LRU). Программный TLB распространён в микропроцессорах для встраиваемых систем с архитектурами MIPS, PowerPC, SPARC и некоторых RISC-V.

2.3 Таблицы страниц

Довольно много современных платформ помимо таблиц TLB, быстрых, но весьма ограниченных по размеру, используют таблицы страниц [7], которые в принципе могут описать отображение всей виртуальной памяти и не имеют ограничений на количество записей.

Для того, чтобы описать таблицу страниц, в виртуальной памяти выделяются несколько допустимых размеров двоичных страниц, которые разрешается отображать в физическую память. Из свойства иерархичности следует, что такая система страниц имеет внутреннюю иерархию и может быть разбита на уровни. В современных архитектурах обычно используются два-три уровня, например, максимальные страницы размера 1 ГБ, страницы второго уровня размера 2 МБ и страницы нижнего уровня размера 4 КБ.

Верхний уровень включает все максимальные страницы, на которые разбивается виртуальное пространство. Для каждой такой страницы имеется запись в таблице страниц, которая либо пуста, либо описывает отображение страницы в физическую память, либо содержит ссылку на описание таблицы более низкого уровня. На нижнем уровне записи могут быть либо пустыми, либо отображающими, ссылок на таблицы на этом уровне быть не может. На верхнем уровне таблиц также иногда может быть установлен запрет на отображающие записи. Пример иерархии таблиц страниц представлен на рис. 2.

Таблицы страниц хранятся в оперативной памяти и обращение к ним гораздо медленнее, чем работа с таблицами TLB. Кроме того, они занимают довольно много места, иногда это не соответствует реально используемой памяти (если используемая память сильно разрежена).

Во многих архитектурах поддерживающих таблицы страниц есть и таблицы TLB, однако, эти таблицы TLB часто имеют только автоматическую аппаратную настройку и их нельзя настраивать программно. Далее, говоря о платформах с таблицами страниц, мы будем иметь в виду архитектуры с таблицами страниц и с автоматической настройкой TLB таблиц, а говоря об архитектурах с таблицами TLB – те архитектуры, в которых TLB таблицы настраиваются программно.

2.4 Ограничения на индексы записей

Вообще говоря, индекс отображающей записи в таблице TLB это просто номер этой записи в таблице. Однако, некоторые платформы определяют сложные зависимости между индексом отображающей записи и той двоичной страницей, которую эта запись отображает.

Для примера рассмотрим платформы типа PowerPC e500 [4]. На данных платформах имеется две таблицы TLB. Одна, называемая TLB0, размера 512 записей с фиксированным размером записи 4 КБ. Вторая, TLB1, значительно меньшего размера (16 или 64 записей), содержит записи, размер которых может быть любым из допустимого множества размеров. Схема устройства таблиц TLB0 и TLB1 приведена на рис. 3.

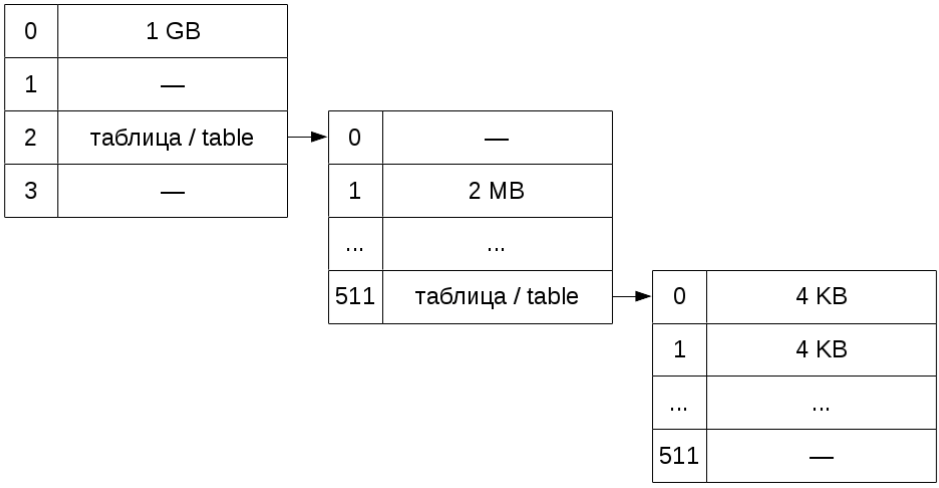


Рис. 2. Пример иерархии таблиц страниц.
Прочерк означает пустую запись, для отображающих записей указан только размер
Fig. 2. Example of page table structure.
Empty entries are marked by '—', map entries are marked by its sizes

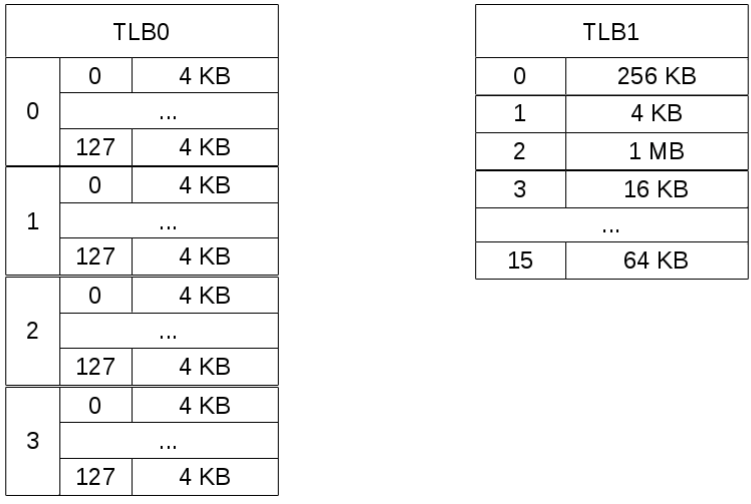
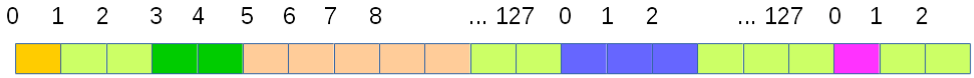


Рис. 3. Таблицы TLB0 и TLB1
Fig. 3. TLB0 and TLB1 structure

В таблице TLB0 записи организованы в 4 канала по 128 записей, индекс записи в канале вычисляется по виртуальному адресу отображаемой страницы: у подряд идущих записей индексы изменяются циклически (0, 1, ..., 127, 0, 1, ..., 127 и т. д.), так как это значения определенной группы битов. Таким образом, страниц с конкретным индексом много больше, чем имеется каналов. Поэтому, при заполнении всех четырех каналов по какому-то индексу, страницы с таким индексом становятся недоступными для отображения.

На рис. 4 приведен пример того, как могут быть задействованы индексы записей из таблицы TLB0. На этом примере видно, что неудачное размещение записей в виртуальной памяти может привести к невозможности отображения записями этой таблицы длинных кусков виртуальной памяти. Так, в данном примере, можно еще отобразить непрерывный кусок длиной 255×4КБ, но нельзя отобразить непрерывный кусок длиной 300×4КБ, хотя формально записей на него хватает, так как в таком куске должно быть не меньше двух страниц с индексом 0, а у нас у этого индекса свободен только один канал.



0	1	2	3	4	5	6	7	8	9	10	11	...	127
0	1	2	3	4	5	6	7	8	9	10	11	...	127
0	1	2	3	4	5	6	7	8	9	10	11	...	127
0	1	2	3	4	5	6	7	8	9	10	11	...	127

Рис. 4. Отображаемые страницы записей из таблицы TLB0 в виртуальной памяти. Указаны используемые индексы. Нижняя таблица показывает загруженность каждого канала. Индекс 0 является самым используемым, у него свободен только один канал.
Fig. 4. Mapped virtual pages for TLB0-entries in the virtual memory. Used indices are specified. At the lower table channel busyness is shown. Zero index is the most used, it has only one free channel

Подобного рода ограничения на индексы имеют и другие платформы PowerPC [5], где индексы записей могут зависеть не только от виртуального адреса, но и от других параметров.

С расположением отображающих записей в виртуальной памяти связаны другие варианты ограничений. Так, на платформах типа MIPS встречается следующая конструкция таблицы TLB [6]. Таблица одна и содержит малое количество записей (например, 16, 24 или 64), но каждая запись является двойной. То есть, по сути, каждая запись таблицы содержит две отображающих записи. При этом допускается, чтобы для одной из половин отображение не было определено.

По параметрам отображения (политике кэширования, правам доступа, адресу страничного блока в физической памяти) две отображающие записи-половины не зависят друг от друга, но в виртуальной памяти они должны следовать строго друг за другом и быть одного размера.

Кроме того, виртуальный адрес первой записи должен быть кратен двойному размеру, то есть такая двойная запись описывает отображение одной двоичной страницы двумя независимыми половинами. В физической памяти образы отображаемых страниц должны иметь тот же индекс половины, что и в виртуальной памяти, то есть, например, физический адрес образа первой половины должен быть также кратен двойному размеру полузаписи, как и виртуальный адрес отображаемой страницы, а физический адрес второй половины, наоборот, кратен её размеру, но не кратен двойному размеру записи (см. рис. 5).

2.5 Выводы

Управление памятью на современных платформах имеет много особенностей и ограничений, таким образом, статическое распределение памяти для современных платформ является нетривиальной задачей, отличающейся от классической задачи выделения памяти.

Тем не менее, организация управления памятью на разных платформах имеет много общего: использование иерархии двоичных страниц, наличие виртуальной памяти, организация отображения виртуальной памяти в физическую с помощью отображающих записей,

общность строения отображающих записей, похожие ограничения на индексы записей и т. д. Это позволяет предложить общую схему работы с различными платформами.

TLB		
0	256 KB	256 KB
1	4 KB	4 KB
2	—	1 MB
3	16 KB	16 KB
...		
15	64 KB	—

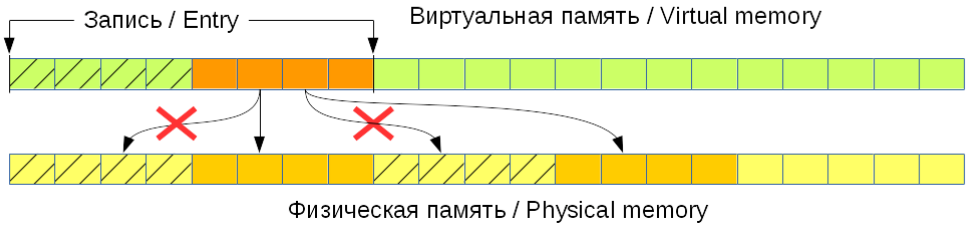


Рис. 5. Платформы MIPS. Таблицы с двойными записями
Fig. 5. Double entry TLB tables (MIPS)

3. Построение инструмента статической раскладки памяти

3.1 Постановка задачи

Формализуем задачу статического распределения памяти. Как мы уже говорили, будем предполагать, что целевая платформа использует виртуальную память.

Также, будем предполагать, что программный комплекс состоит из нескольких модулей. В каждом программном модуле имеется ядро и несколько пользовательских разделов.

Для нужд ядра модуля выделяется часть виртуальной памяти, которая называется привилегированной, для нужд разделов выделяется часть виртуальной памяти, называемая непривилегированной. Разделение виртуальной памяти на привилегированную и непривилегированную части может быть реализовано и аппаратными средствами, но может быть и только логическим, для целей раскладки это не важно.

Мы будем считать, что в каждый конкретный момент времени активирован один модуль (его ядро) и один пользовательский раздел. Состояние виртуальной памяти будет задаваться множеством адресов, которые могут быть транслированы в данный момент и то, куда они транслируются, то есть, по сути, активными в данный момент отображающими записями. При переключении одного раздела на другой меняется состояние виртуальной памяти.

Доступные виртуальные адреса для любых двух ядер (или пользовательских разделов) одинаковые, но отображающие записи будут у каждого ядра (раздела) свои, так что для каждого раздела или ядра можно определить его личное (или частное) виртуальное пространство, в котором будут доступны только нужные этому разделу или ядру отображающие записи.

Совокупность частных виртуальных пространств ядра модуля и некоторого пользовательского раздела этого модуля будем называть полным виртуальным пространством. Все ограничения по количеству доступных отображающих записей всегда касаются именно полного виртуального пространства, так как отображающие записи в равной степени доступны и ядру, и пользовательскому разделу.

Для всех модулей и разделов определено единое пространство оперативной физической памяти, а также единое пространство физических адресов устройств. Для выполнения временных ограничений и сохранения предсказуемости времени доступа к памяти необходимо отказаться от подкачки, поэтому мы считаем, что все ядра и разделы должны находиться в физической памяти всегда.

Разработчик программного обеспечения должен определить требования на память для всех разрабатываемых модулей — их ядер и пользовательских разделов. Для описания требований на память мы используем старую идею логических сегментов, но в новом качестве.

Требования на память со стороны программных модулей организованы в *блоки памяти*. *Блок памяти* — это непрерывный участок в виртуальной памяти, для которого указаны правила отображения его в физическую память. Блоки памяти также делятся на блоки памяти ядра и блоки памяти пользовательских разделов.

Каждый блок памяти имеет атрибуты, существенные для раскладки памяти:

- размер блока;
- адрес блока в виртуальной памяти (может быть не определен);
- адрес блока в физической памяти (может быть не определен);
- права доступа;
- политика кэширования;
- коэффициент выравнивания;
- является ли блок непрерывным в физической памяти;
- размер зоны безопасности до блока (передняя зона безопасности);
- является ли передняя зона безопасности неотображаемой;
- размер зоны безопасности после блока (задняя зона безопасности);
- является ли задняя зона безопасности неотображаемой.

У блоков памяти могут быть и другие атрибуты, влияющие на раскладку памяти.

В качестве блоков памяти разработчик может выделять разные вспомогательные структуры (буфера, стеки, кучи и т.п.), области данных, области, в которых содержатся исполняемые инструкции, области, служащие для общения разделов и ядер (разделяемая память) и т. д.

Уровень детализации разделения на блоки памяти определяет сам разработчик. Размеры блоков памяти никак не привязаны к структуре двоичных страниц (как это обычно бывает при тривиальном подходе) и диктуются только нуждами проектируемого программного обеспечения. Задача раскладки теперь формулируется следующим образом.

Необходимо определить виртуальные адреса для каждого блока и регионы физической памяти, в которые отображается данный блок, а также систему отображающих записей, соответствующую конкретной архитектуре устройства управления памятью, которая будет отображать блоки памяти из виртуальной памяти в физическую в точном соответствии с описанием этих блоков.

3.2 Общая схема построения статической раскладки памяти

В данном разделе мы рамочно описываем принципы построения статической раскладки памяти, поскольку формат статьи не позволяет описать подробности реализации. Тем не менее, мы надеемся углубить и расширить данное описание в последующих статьях.

Введем понятие *преформы* – модели последовательности отображающих записей, непрерывной в виртуальной памяти. Концепция преформы олицетворяет связь между виртуальной и физической памятью, хотя в начале эта связь может быть еще не определена. Преформа как бы существует сразу и в виртуальной, и в физической памяти. Мы будем покрывать преформами блоки памяти, определенные разработчиком.

Внутренняя структура преформы не произвольна, а должна быть согласована с иерархией двоичных страниц. Так, например, невозможна преформа, состоящая из записей размера 16, 4, 16 в указанном порядке, так как иерархия двоичных страниц требует, чтобы между страницами размера 16 находился промежуток длины, кратной 16, а 4 на 16 не делится.

Если покрываемый блок изначально фиксирован в виртуальной памяти, то это также накладывает ограничения на структуру покрывающей преформы и на расположение блока внутри неё (см. рис. 6), а, кроме того, предписывает преформе определенный адрес в виртуальной памяти. О других ограничениях будет сказано ниже.

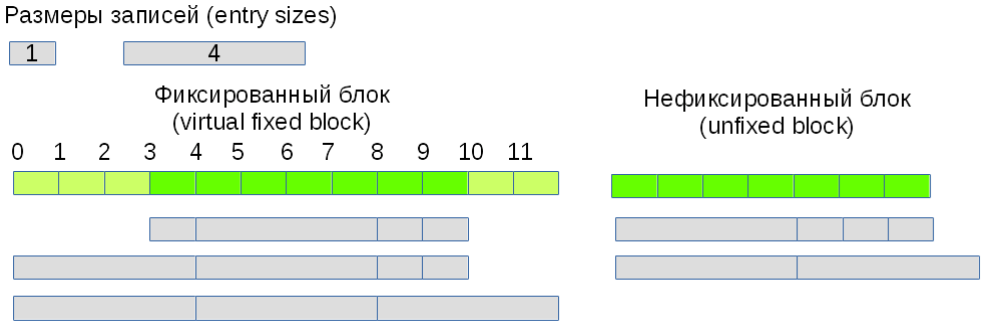


Рис. 6. Построение преформ, покрывающих блоки
Приведен пример фиксированного в виртуальной памяти блока и нефиксированного блока
Fig. 6. Entry list preforms for memory blocks.
For example, case of virtual fixed memory block and case of unfixed memory block

Для того, чтобы покрыть блок памяти преформой, нужно указать

- к какому частному виртуальному пространству относится покрывающая преформа – это частное виртуальное пространство совпадает с частным виртуальным пространством блока;
- атрибуты отображающих записей – права доступа и политику кэширования, они так же должны совпадать с соотвествующими атрибутами покрываемого блока;
- последовательность отображающих записей, из которых состоит преформа (с указанием их размеров);
- относительный адрес покрываемого блока внутри преформы.

При наличии системы покрывающих преформ, если удастся решить задачу размещения преформ в каждом полном виртуальном пространстве и в физической памяти, то такое размещение автоматически создает систему отображающих записей — это просто все записи преформ, относящихся к данному полному виртуальному пространству. Работа с системой преформ (а не с отдельными записями) позволяет регулировать общие характеристики, такие как количество используемых записей или количество используемой памяти. Двойственная природа преформ (виртуальная и физическая), дает необходимую гибкость при размещении их в памяти (см. рис. 7).

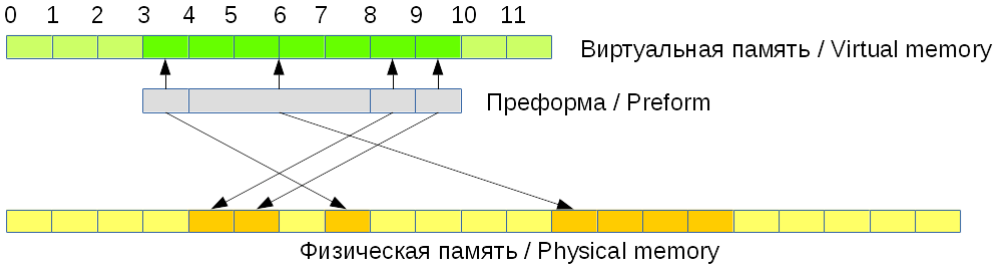


Рис. 7. Размещение преформы в виртуальной и физической памяти
Fig. 7. Placing the preform in the virtual memory and in the physical memory

Итак, наша схема алгоритма построения статической раскладки памяти будет включать следующие шаги:

- построение системы покрывающих преформ, обладающих подходящими характеристиками;
- упорядочивание преформ и/или их записей для последующего размещения в памяти;
- размещение построенных преформ в виртуальной и физической памяти;
- корректировка системы преформ при возникновении проблем с размещением.

Отдельно может проводиться анализ того, возможно ли в принципе построение раскладки для данной системы блоков памяти, а также поиск проблемных мест в системе требований на память (например, тотальная нехватка оперативной памяти или невозможность построения раскладки в силу недостаточного количества доступных записей).

Рассмотрим каждый шаг алгоритма отдельно.

При построении системы преформ должны учитываться

- размеры и количество доступных отображающих записей для каждого полного виртуального пространства;
- количество доступной физической памяти;
- ограничения на индексы записей, зависящие от особенностей используемой аппаратуры;
- ограничения, возникающие из внутренних атрибутов блоков, таких, как неотображаемые зоны безопасности;
- возможность покрытия нескольких блоков одной записью, если атрибуты блоков это позволяют;
- другие ограничения, возникающие из особенностей аппаратуры или специфики задачи.

Упорядочивание системы преформ имеет важное значение для успешного решения задачи раскладки. Могут быть приняты разные подходы к порядку размещения, но несомненно все они должны учитывать

- размер самой преформы, так как в виртуальной памяти под нее необходимо выделять непрерывный кусок;
- размеры записей, входящих в преформу, так как размещение малой записи в большой странице может помешать разместить большую запись;
- фиксированность блоков преформы в виртуальной и/или физической памяти;
- другие ограничения (например, коэффициенты выравнивания блоков).

Размещение одной и той же преформы в виртуальной и физической памяти позволяет задать отображения для записей, входящих в преформу. Размещение преформ в памяти должно производиться в определенном порядке с учетом

- семантических требований корректности раскладки, таких, как запрет пересечений записей в виртуальной памяти или запрет пересечения блоков памяти в физической памяти, или разрешение пересечений зон безопасности в виртуальной памяти и т.п.
- размера размещаемых записей (адрес записи пропорционален ее размеру);
- изначальной фиксации покрытых блоков в виртуальной и/или физической памяти или отсутствии такой фиксации;
- допустимости раздельного размещения записей одной преформы в физической памяти;
- наличия семантических связей между положением страницы, отображаемой записью, в виртуальной памяти и положением ее образа в физической памяти;
- наличия дополнительных требований.

Коррекция системы преформ может состоять в

- уплотнении преформ, уже находящихся в памяти, либо слиянии преформ вне памяти с преформами в памяти;
- разделении преформы на более мелкие, если имеются ресурсы отображающих записей подходящего размера;
- повторном построении системы преформ с учетом информации, полученной при первичном построении;
- другие методы, более специфичные для конкретных архитектур.

Если удалось корректно разместить покрывающие преформы в виртуальной и физической памяти, то для каждой записи, входящей в преформу определены виртуальный и физический адрес. Для блоков, покрываемых преформой, также определены виртуальный адрес и система регионов физической памяти, в которые отображается блок. Таким образом, определены и расположение блоков, и таблицы настройки устройства управления памятью.

Итак, ***корректное размещение системы преформ в виртуальной и физической памяти дает решение задачи статической раскладки памяти.***

4. Практические результаты

В рамках разработки ОСРВ на базе стандарта ARINC 653 нами была реализована система построения статической раскладки памяти, основанная на приведенных выше принципах, для двух десятков видов платформ, в которые входят:

- платформы типа PowerPC (e500 и 470 модификаций);
- платформы, использующие таблицы страниц (i.MX 6, AArch64 Cortex-A55, RISC-V RV32I, и др.);
- платформы типа MIPS.

В настоящее время список платформ расширяется.

Временные характеристики работы ОСРВ, использующей полученную раскладку памяти, приведены в работе [9], а здесь мы приведем статистику, касающуюся собственно раскладки памяти.

Ниже представлены некоторые количественные характеристики конфигураций памяти, для которых строилась раскладка памяти:

- количество модулей в одной конфигурации: до 4;
- количество разделов в одном модуле: до 32;

- количество групп блоков разделяемой памяти: до 14;
- количество блоков с ненулевыми зонами безопасности: до 20.

Общее число использованных конфигураций памяти превышало 300000. Для всех использованных конфигураций была построена корректная раскладка памяти. Проверка корректности производилась автоматически, с помощью разработанного нами инструмента.

Ниже приведена статистика по общему количеству блоков и отображающих записей, а также по времени раскладки. Она получена на репрезентативной выборке рабочих конфигураций памяти из примерно 10000 конфигураций. Для того, чтобы не перегружать графики, конфигурации памяти разделены на две группы по типу целевой платформы: конфигурации памяти для платформ, использующих программно настраиваемые таблицы TLB (Power PC e500mc, MIPS 24Kf и др.), и конфигурации памяти для платформ, использующих таблицы страниц, в которых таблицы TLB настраиваются автоматически (AArch64 Cortex-A55, RISC-V RV32I и др.).

На рис. 8 показано общее количество блоков памяти в конфигурациях памяти: по горизонтальной оси отмечено количество блоков в конфигурации, по вертикальной — количество конфигураций в выборке с таким количеством блоков памяти.

На рис. 9 показано общее количество отображающих записей, использованных для отображения блоков памяти: по горизонтальной оси отмечено количество использованных записей в раскладке памяти, по вертикальной — количество конфигураций памяти в выборке с таким количеством отображающих записей в раскладке памяти. Для платформ с программно настраиваемыми таблицами TLB указано количество отображающих записей в таблицах TLB, для платформ с таблицами страниц и аппаратно настраиваемыми таблицами TLB указано количество отображающих записей в таблицах страниц, так как только эти таблицы можно настраивать программно. Различие в количествах отображающих записей для двух видов платформ обусловлено тем, что

- количество записей в таблицах TLB ограничено по сравнению с таблицами страниц, которые могут свободно покрывать всю имеющуюся память;
- в таблицах страниц обычно имеется меньше доступных размеров для отображающих записей, чем в таблицах TLB, и из-за этого между соседними размерами имеется значительный разрыв (например, 4КБ и 4МБ), что приводит к увеличению количества используемых записей;
- в данный момент раскладчик настроен на минимальный перерасход памяти и, таким образом, для покрытия блоков памяти конфигураций памяти на платформах с таблицами страниц используется много мелких записей.

Тем не менее, для платформ с таблицами страниц возможна оптимизация раскладки по количеству используемых записей. В перспективе нами планируется исследовать возможности ее реализации. Это тем более представляет интерес, поскольку так можно повлиять на настройку таблиц TLB, которые на данных платформах настраиваются аппаратно, и снизить число изменений содержания записей этих таблиц.

На рис. 10 показано среднее время построения статической раскладки памяти для различных платформ. Как и раньше, платформы разделены на две группы, принадлежность к которым показана цветом, время указано в секундах. Время раскладки для всех конфигураций памяти не превышает нескольких секунд, что для этапа статической настройки системы является вполне удовлетворительным.

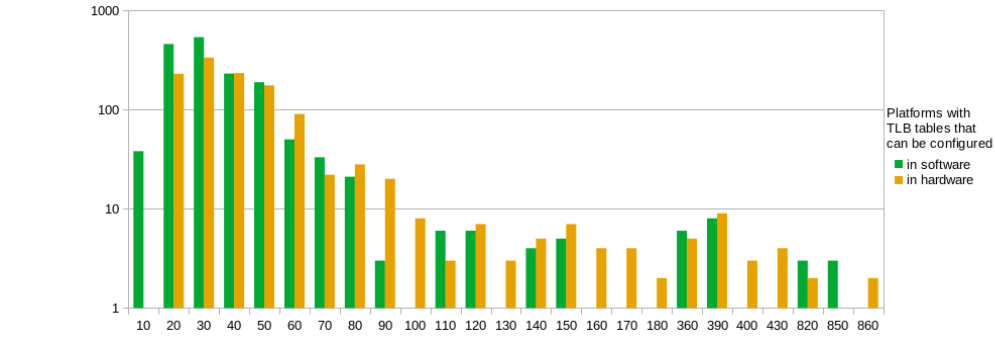


Рис. 8. Количество блоков в конфигурациях памяти для платформ с программно настраиваемыми таблицами TLB и для платформ с таблицами страниц и аппаратно настраиваемыми таблицами TLB
Fig. 8. Number of memory blocks in memory configurations for platforms with software configured TLB tables and platforms with page tables and hardware configured TLB tables

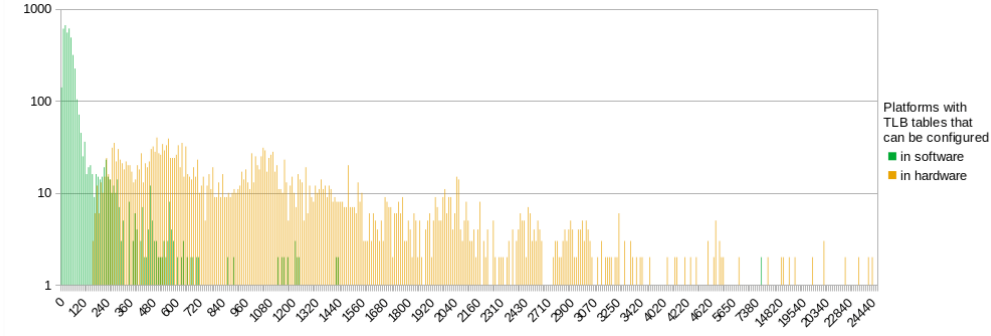


Рис. 9. Количество использованных отображающих записей в раскладке памяти для конфигураций памяти для платформ с таблицами TLB и для платформ с таблицами страниц
Fig. 9. Number of table entries in found MMU configurations for platforms with software configured TLB tables and platforms with page tables and hardware configured TLB tables

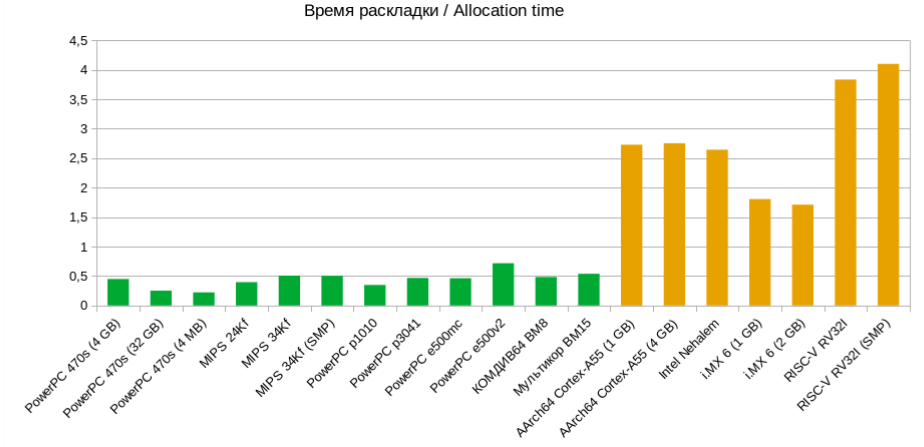


Рис. 10. Среднее время построения статической раскладки памяти для различных платформ (в секундах)
Fig. 10. Memory layout creation time for different platforms (in sec)

Разница во времени для платформ с программно настраиваемыми таблицами TLB и платформ с таблицами страниц и аппаратно настраиваемыми таблицами TLB коррелирует с общим

количеством использованных записей. Здесь нужно отметить, что первоочередной задачей было добиться собственно построения раскладки памяти, удовлетворяющей требованиям разработчиков ПО. Тем не менее, в перспективе планируется оптимизация процесса раскладки памяти и по времени тоже.

Наши результаты показывают, что построение статической раскладки памяти для направленных на повышение безопасности работы ОСРВ и сложно устроенных требований на память, не только возможно, но возможно за приемлемое время. Таким образом, поиск решения задачи построения статической раскладки памяти для удовлетворения такого рода требований является перспективным и нужным направлением исследований.

5. Заключение

Статическая раскладка памяти, полученная с помощью наших инструментов, была опробована в ОСРВ на базе ARINC 653, разработанной в рамках методологии построения ОСРВ на базе ARINC 653, развиваемой в ИСП РАН [8-9].

Апробация метода показала его работоспособность, эффективность и адаптивность к разным условиям.

Автор благодарит команду разработчиков ОСРВ в ИСП РАН и в особенности В.Ю. Чепцова за постановку задачи, плодотворные обсуждения и ценные замечания, без которых данная работа не могла бы состояться.

Список литературы / References

[1]. Таненбаум Э., Бос Х. Современные операционные системы. СПб., Питер, 2019, 1120 с.
[2]. Couler J. F., Glaser E.L. Shared-access Data Processing System. Patent 3412382, November 1968.
[3]. 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
[4]. e500mcRM revision 3. e500mc Core Reference Manual. Freescale Semiconductor, Inc. 2013-03. 440 p. https://www.nxp.com/docs/en/reference-manual/E500MCRM.pdf [ppc500]
[5]. PowerPC 476FP Embedded Processor Core User's Manual. International Business Machines Corporation. 2014-01-10. 320 p.
[6]. 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
[7]. 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
[8]. 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.
[9]. Khoroshilov A.V., Cheptsov V.Y. Robust Resource Partitioning Approach for ARINC 653 RTOS [9]

Информация об авторах / Information about authors

Софья Анатольевна ЗЕЛЕНОВА – кандидат физико-математических наук, научный сотрудник ИСП РАН. Научные интересы: алгоритмы статического распределения ресурсов, теория расписаний, операционные системы реального времени, тестирование на основе моделей, теория чисел.

Sofya Anatolyevna ZELENOVA – Cand. Sci.Ph. (Phys.-Math.), Researcher at ISP RAS. Research interests: static resource allocation algorithms, scheduling algorithms, real-time operating systems, model-based testing, number theory.