Операционные системы

Основы управления памятью

Классификация методов распределения памяти

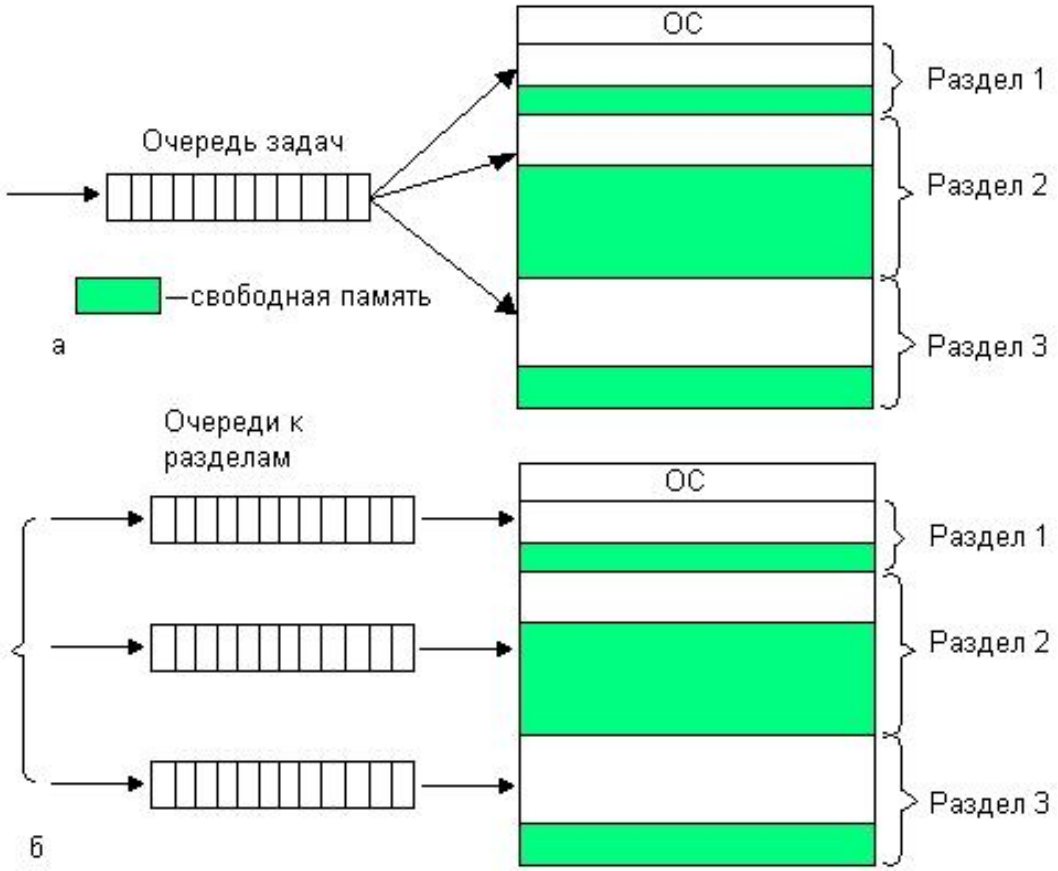
Основы управления памятью

Методы распределения памяти без использования дискового пространства

Распределение памяти фиксированными разделами

- Самым простым способом управления оперативной памятью является разделение ее на несколько разделов фиксированной величины. Это может быть выполнено вручную оператором во время старта системы или во время ее генерации.
- Очередная задача, поступившая на выполнение, помещается либо в общую очередь (рисунок а), либо в очередь к некоторому разделу (рисунок б).
- Подсистема управления памятью в этом случае выполняет следующие задачи:
  - сравнивая размер программы, поступившей на выполнение, и свободных разделов, выбирает подходящий раздел,
  - осуществляет загрузку программы и настройку адресов.

Распределение памяти фиксированными разделами



Эта схема была реализована в IBM OS/360, DEC RSX-11 и ряде других систем.

а - с общей очередью;
б - с отдельными очередями

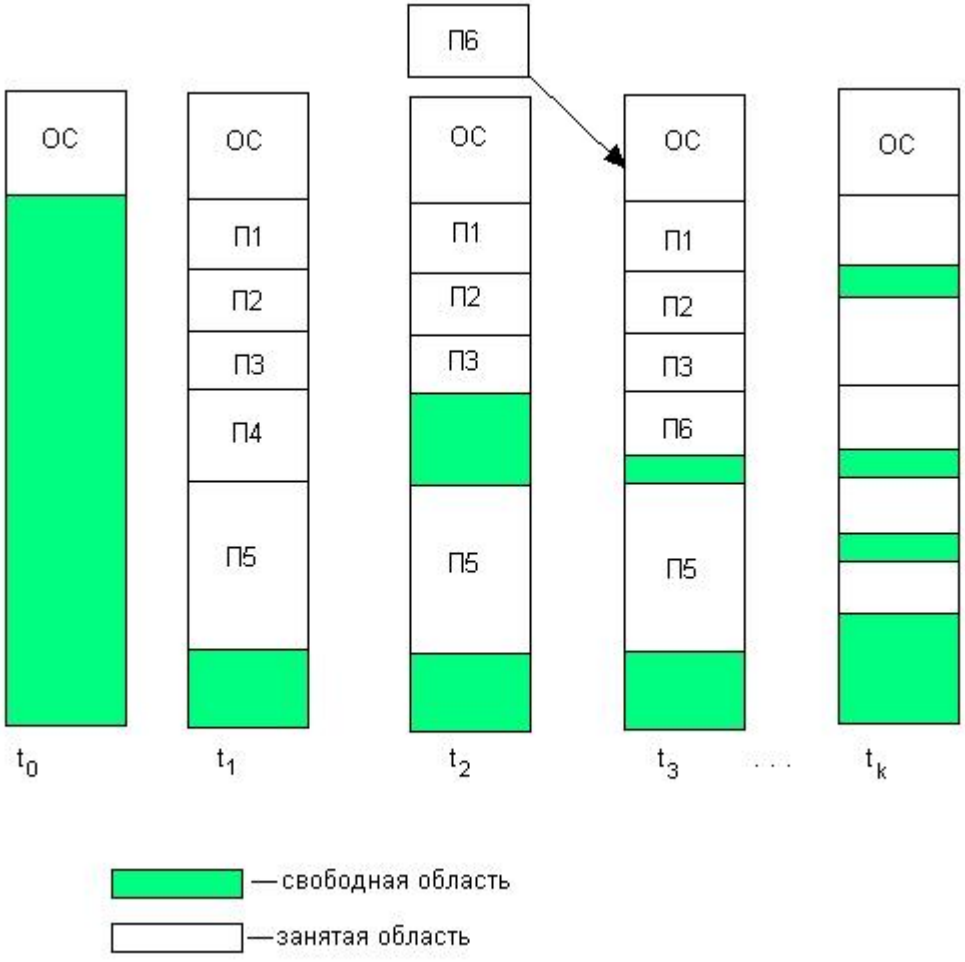
Распределение памяти фиксированными разделами

- При очевидном преимуществе – простоте реализации, данный метод имеет существенный недостаток – жесткость. Так как в каждом разделе может выполняться только одна программа, то уровень мультипрограммирования заранее ограничен числом разделов не зависимо от того, какой размер имеют программы.
- Другим существенным недостатком является то, что предлагаемая схема сильно страдает от ***внутренней фрагментации*** – потери части памяти, выделенной процессу, но не используемой им. *Фрагментация* возникает потому, что процесс не полностью занимает выделенный ему раздел или потому, что некоторые разделы слишком малы для выполняемых пользовательских программ.

Распределение памяти разделами переменной величины

- В этом случае память машины не делится заранее на разделы. Сначала вся память свободна. Каждой вновь поступающей задаче выделяется необходимая ей память. Если достаточный объем памяти отсутствует, то задача не принимается на выполнение и стоит в очереди. После завершения задачи память освобождается, и на это место может быть загружена другая задача. Таким образом, в произвольный момент времени оперативная память представляет собой случайную последовательность занятых и свободных участков (разделов) произвольного размера.
- На рисунке показано состояние памяти в различные моменты времени при использовании динамического распределения. Так в момент t_0 в памяти находится только ОС, а к моменту t_1 память разделена между 5 задачами, причем задача П4, завершаясь, покидает память. На освободившееся после задачи П4 место загружается задача П6, поступившая в момент t_3.

Распределение памяти разделами переменной величины



Распределение памяти разделами переменной величины

- Задачами операционной системы при реализации данного метода управления памятью является:
  - ведение таблиц свободных и занятых областей, в которых указываются начальные адреса и размеры участков памяти,
  - при поступлении новой задачи - анализ запроса, просмотр таблицы свободных областей и выбор раздела, размер которого достаточен для размещения поступившей задачи,
  - загрузка задачи в выделенный ей раздел (настройка адресов) и корректировка таблиц свободных и занятых областей,
  - после завершения задачи корректировка таблиц свободных и занятых областей.
- Программный код не перемещается во время выполнения, то есть может быть проведена единовременная настройка адресов посредством использования перемещающего загрузчика.

Стратегии выбора раздела переменной величины

- В какой раздел помещать процесс? Наиболее распространены три стратегии.
  - Стратегия первого подходящего (First fit). Процесс помещается в первый подходящий по размеру раздел.
  - Стратегия наиболее подходящего (Best fit). Процесс помещается в тот раздел, где после его загрузки останется меньше всего свободного места.
  - Стратегия наименее подходящего (Worst fit). При помещении в самый большой раздел в нем остается достаточно места для возможного размещения еще одного процесса.
- Моделирование показало, что доля полезно используемой памяти в первых двух случаях больше, при этом первый способ несколько быстрее. Попутно заметим, что перечисленные стратегии широко применяются и другими компонентами ОС, например для размещения файлов на диске.

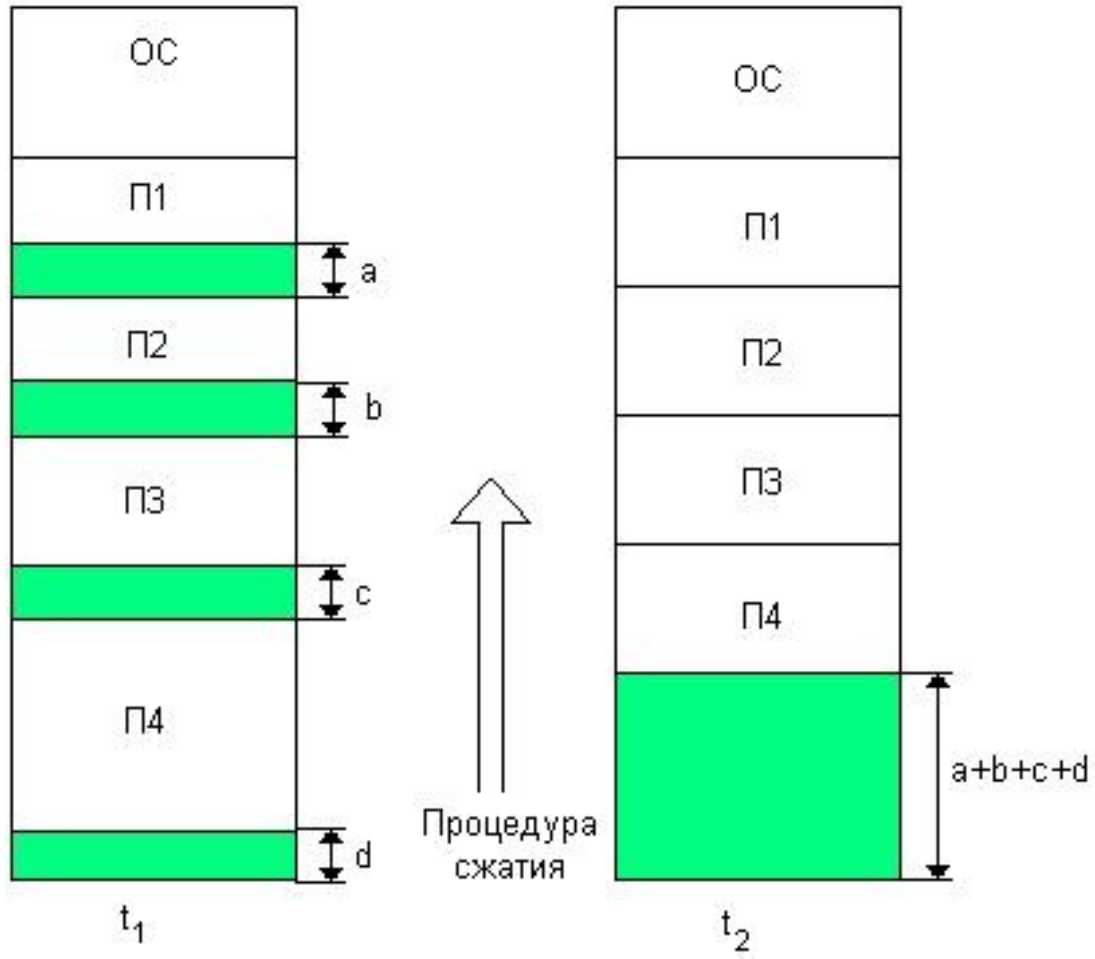
Распределение памяти разделами переменной величины

- По сравнению с методом распределения памяти фиксированными разделами данный метод обладает гораздо большей гибкостью, но ему присущ очень серьезный недостаток – *фрагментация памяти*.
- Фрагментация в данном случае – это наличие большого числа несмежных участков свободной памяти очень маленького размера (фрагментов). Настолько маленького, что ни одна из вновь поступающих программ не может поместиться ни в одном из участков, хотя суммарный объем фрагментов может составить значительную величину, намного превышающую требуемый объем памяти.

Перемещаемые разделы

- Одним из методов борьбы с фрагментацией является перемещение всех занятых участков в сторону старших либо в сторону младших адресов, так, чтобы вся свободная память образовывала единую свободную область (слайд). В дополнение к функциям, которые выполняет ОС при распределении памяти переменными разделами, в данном случае она должна еще время от времени копировать содержимое разделов из одного места памяти в другое, корректируя таблицы свободных и занятых областей. Эта процедура называется "сжатием".
- Сжатие может выполняться либо при каждом завершении задачи, либо только тогда, когда для вновь поступившей задачи нет свободного раздела достаточного размера. В первом случае требуется меньше вычислительной работы при корректировке таблиц, а во втором – реже выполняется процедура сжатия.
- Так как программы перемещаются по оперативной памяти в ходе своего выполнения, то преобразование адресов должно выполняться динамическим способом.
- Хотя процедура сжатия и приводит к более эффективному использованию памяти, она может потребовать значительного времени, что часто перевешивает преимущества данного метода.

Перемещаемые разделы



Основы управления памятью

Методы распределения памяти с использованием дискового пространства

Понятие виртуальной памяти

- Уже достаточно давно пользователи столкнулись с проблемой размещения в памяти программ, размер которых превышал имеющуюся в наличии свободную память. Решением было разбиение программы на части, называемые *оверлеями*. 0-ой оверлей начинал выполняться первым. Когда он заканчивал свое выполнение, он вызывал другой оверлей. Все оверлеи хранились на диске и перемещались между памятью и диском средствами ОС. Однако разбиение программы на части и планирование их загрузки в ОП должен был осуществлять программист.
- Развитие методов организации вычислительного процесса в этом направлении привело к появлению метода, известного под названием *виртуальная память*.
- Термин *виртуальная память* обычно ассоциируется с возможностью адресовать пространство памяти, гораздо большее, чем емкость первичной (реальной) памяти конкретной вычислительной машины. Концепция виртуальной памяти является далеко не новой, впервые она была реализована в вычислительной машине Atlas, созданной в Манчестерском университете в Англии в 1960г.

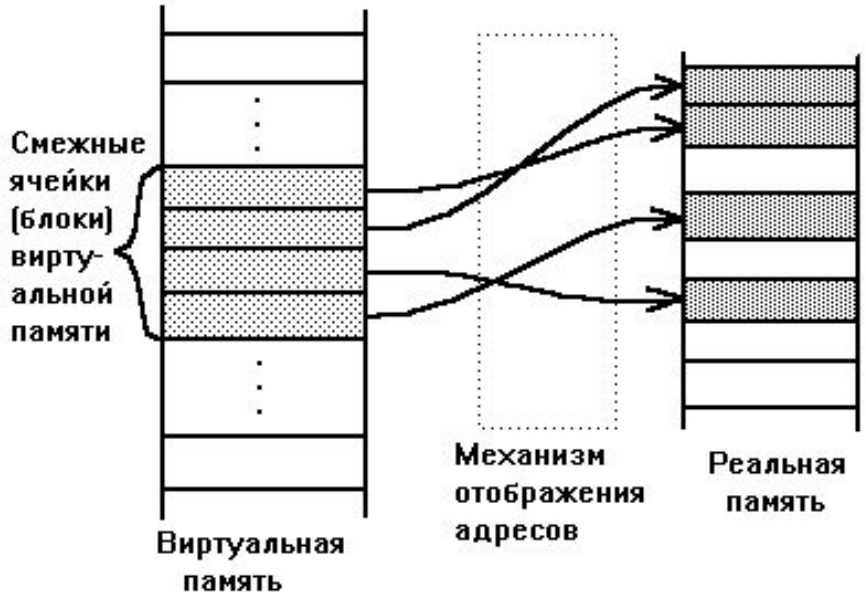
Понятие виртуальной памяти

- Виртуальная память (ВП) – это совокупность программно-аппаратных средств, позволяющих выполнять программы, размер которых превосходит имеющуюся оперативную память.
- Для этого менеджер ВП решает следующие задачи:
  - размещает данные в запоминающих устройствах разного типа, например, часть программы в оперативной памяти, а часть на диске;
  - перемещает по мере необходимости данные между запоминающими устройствами разного типа, например, подгружает нужную часть программы с диска в оперативную память (свопинг);
  - преобразует виртуальные адреса в физические.

Физические и виртуальные адреса

- Суть концепции виртуальной памяти заключается в том, что адреса, к которым обращается выполняющийся процесс, отделяются от адресов, реально существующих в первичной памяти:
  - адреса, на которые делает ссылки выполняющийся процесс, называются *виртуальными адресами (ВА);*
  - адреса, которые существуют в первичной памяти, называются *реальными* (или физическими) *адресами (ФА)*.
- Для установления соответствия между ВА и ФА разработаны различные способы. Так, механизмы *динамического преобразования адресов* (DAT - Dynamic Address Translation) обеспечивают преобразование ВА в ФА время выполнения процесса.

Механизм отображения адресов



- Все подобные системы обладают общим свойством: смежные адреса виртуального адресного пространства процесса не обязательно будут смежными в реальной памяти, это свойство называют “искусственной смежностью”.

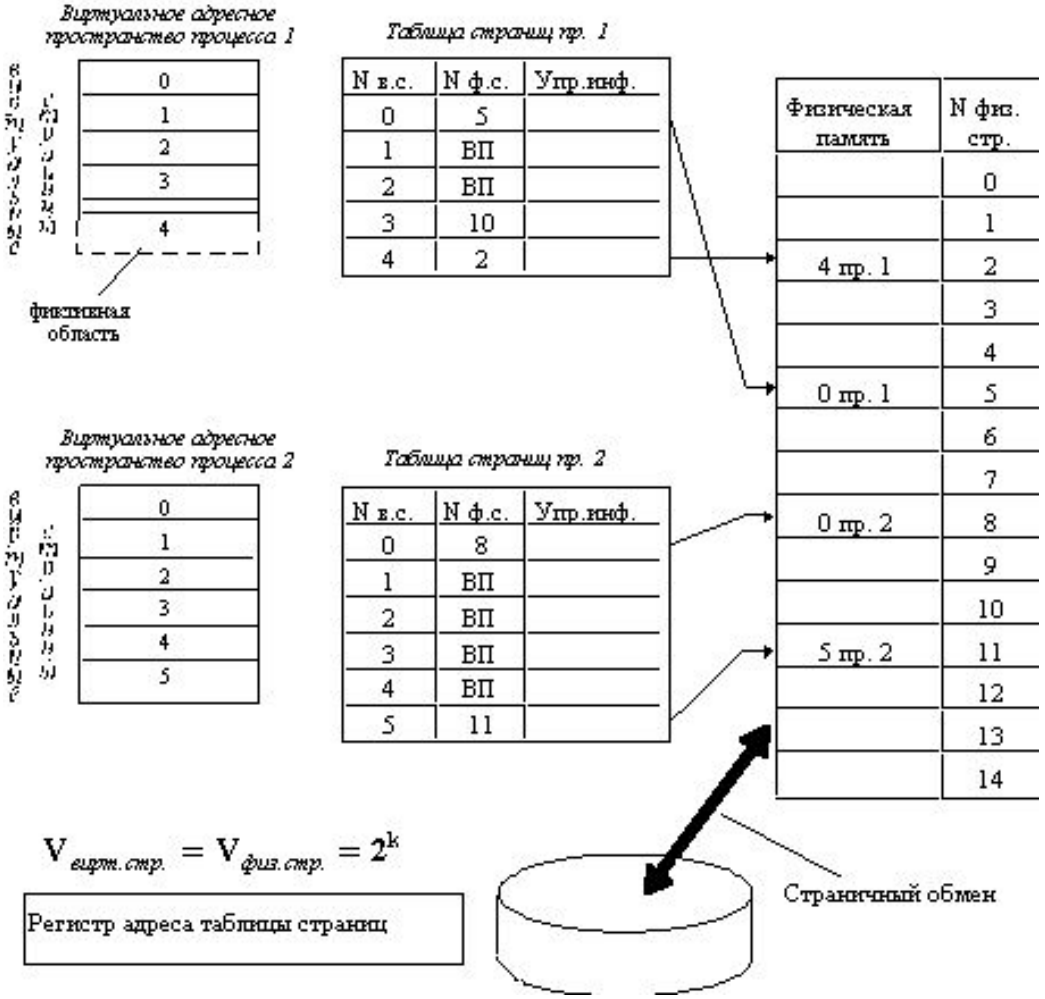
Способы организации виртуальной памятия

- страничное распределение
- сегментное распределение
- сегментно-страничное распределение

Страничное распределение

- Виртуальное адресное пространство (ВАП) каждого процесса делится на части одинакового, фиксированного для данной ОС размера, называемые виртуальными страницами. Размер страницы кратен степени двойки, это позволяет упростить механизм преобразования адресов.
- ОП делится на физические страницы такого же размера.
- При загрузке процесса ОС создает для каждого процесса таблицу страниц, в которой устанавливается соответствие между номерами виртуальных и физических страниц для страниц, загруженных в ОП, или делается отметка о том, что виртуальная страница выгружена на диск.
- При загрузке процесса часть виртуальных страниц процесса помещается в оперативную память, а остальные – на диск.
- При активизации очередного процесса в специальный регистр процессора загружается адрес таблицы страниц данного процесса.

Таблицы страниц процессов



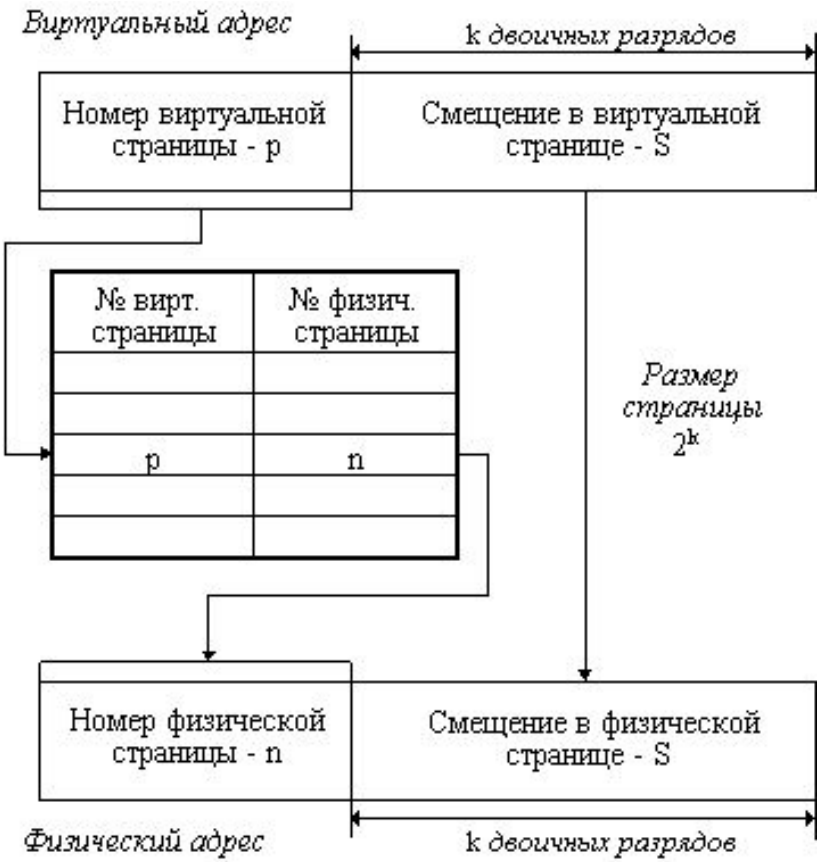
Свопинг при страничном распределении

- При каждом обращении к памяти происходит чтение из таблицы страниц информации о виртуальной странице, к которой произошло обращение.
  - Если данная виртуальная страница находится в ОП, то выполняется преобразование ВА в ФА.
  - Если же нужная виртуальная страница в данный момент выгружена на диск, то происходит так называемое страничное прерывание. Выполняющийся процесс переводится в состояние ожидания, и активизируется другой процесс из очереди готовых. Параллельно программа обработки страничного прерывания находит на диске требуемую виртуальную страницу и пытается загрузить ее в ОП. Если в памяти имеется свободная физическая страница, то загрузка выполняется немедленно, если же свободных страниц нет, то решается вопрос, какую страницу следует выгрузить из ОП.

Страничное распределение: преобразование ВА в ФА

- Рассмотрим механизм преобразования виртуального адреса в физический при страничной организации памяти.
- Виртуальный адрес при страничном распределении может быть представлен в виде пары (p, s), где p - номер виртуальной страницы процесса (нумерация страниц начинается с 0), а s - смещение в пределах виртуальной страницы. Учитывая, что размер страницы равен 2 в степени k, смещение s может быть получено простым отделением k младших разрядов в двоичной записи виртуального адреса. Оставшиеся старшие разряды представляют собой двоичную запись номера страницы p.

Страничное распределение: преобразование ВА в ФА



1. на основании начального адреса таблицы страниц, номера виртуальной страницы и длины записи в таблице страниц определяется адрес нужной записи в таблице,
2. из этой записи извлекается номер физической страницы,
3. к номеру физической страницы присоединяется смещение.

Выбор размера страницы

- при малых страницах:
  - меньшая внутренняя фрагментация страниц и повышается эффективность использования оперативной памяти;
  - снижаются расходы времени на свопинг страниц (так как больше страниц помещаются в памяти);
- при больших страницах:
  - меньшие затраты на поиск и управление страницами (таблицы имеют меньший размер);
  - выше эффективность обмена с внешней памятью.

Достоинства и недостатки страничного распределения

- Использование страниц размером кратным равен степени 2, позволяет применить операцию конкатенации (присоединения) вместо более длительной операции сложения, что уменьшает время получения ФА.
- На производительность системы со страничной организацией памяти влияют временные затраты, связанные с обработкой страничных прерываний и преобразованием ВА в ФА.
- Время преобразования виртуального адреса в физический в значительной степени определяется временем доступа к таблице страниц. В связи с этим таблицу страниц стремятся размещать в "быстрых" запоминающих устройствах. Это может быть, например, набор специальных регистров или память, использующая для уменьшения времени доступа ассоциативный поиск и кэширование данных.

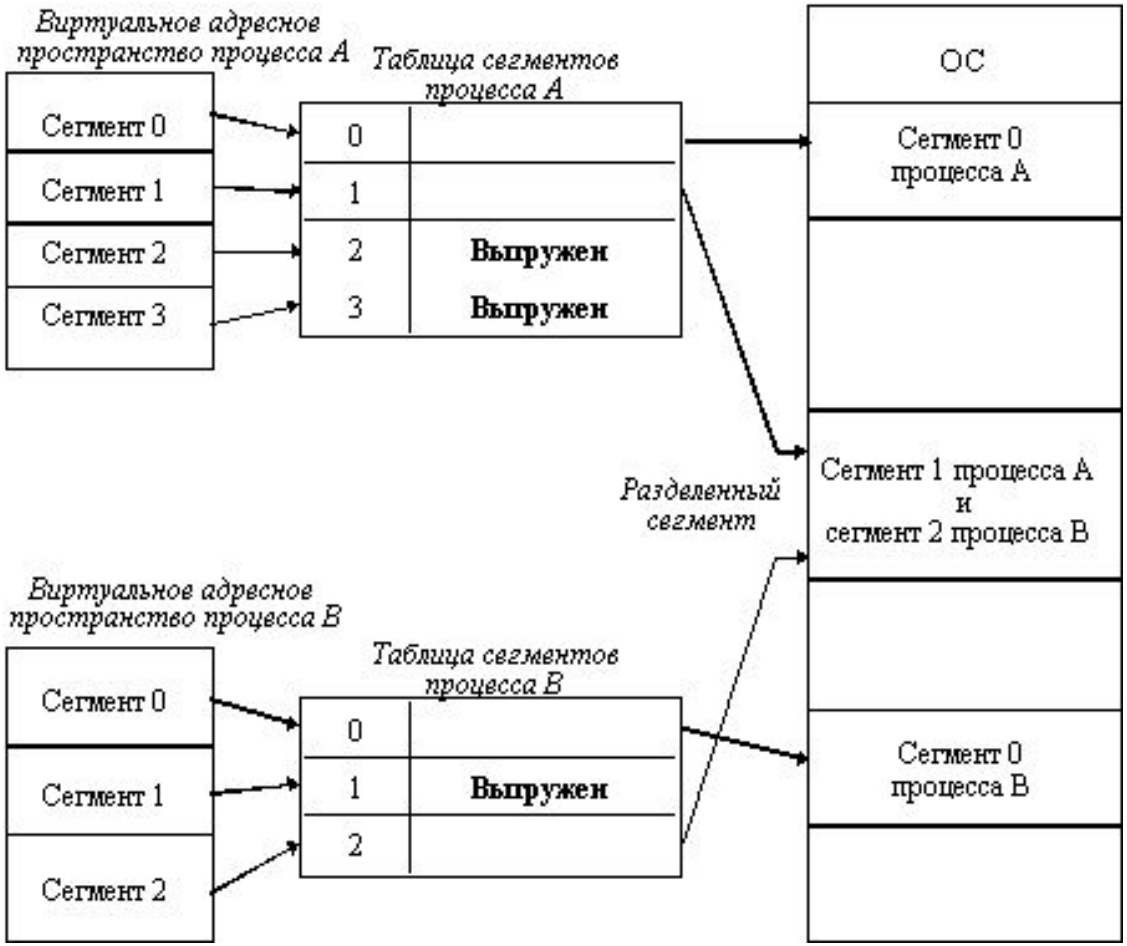
Сегментное распределение

- При страничной организации виртуальное адресное пространство процесса делится механически на равные части. Это не позволяет дифференцировать способы доступа к разным частям программы (сегментам), а это свойство часто бывает очень полезным.
- Например, можно запретить обращаться с операциями записи и чтения в кодовый сегмент программы, а для сегмента данных разрешить только чтение.
- Кроме того, разбиение программы на "осмысленные" части делает принципиально возможным разделение одного сегмента несколькими процессами. Например, если два процесса используют одну и ту же математическую подпрограмму, то в оперативную память может быть загружена только одна копия этой подпрограммы.

Сегментное распределение

- ВАП процесса делится на сегменты, размер которых определяется программистом с учетом смыслового значения содержащейся в них информации. Отдельный сегмент может представлять собой подпрограмму, массив данных и т.п. Иногда сегментация программы выполняется по умолчанию компилятором.
- При загрузке процесса часть сегментов помещается в оперативную память (при этом для каждого из этих сегментов ОС подыскивает подходящий участок свободной памяти), а часть сегментов размещается в дисковой памяти. Сегменты одной программы могут занимать в оперативной памяти несмежные участки.
- Во время загрузки система создает таблицу сегментов процесса (аналогичную таблице страниц), в которой для каждого сегмента указывается начальный физический адрес сегмента в оперативной памяти, размер сегмента, правила доступа, признак модификации, признак обращения к данному сегменту за последний интервал времени и некоторая другая информация. Если ВАП нескольких процессов включают один и тот же сегмент, то в таблицах сегментов этих процессов делаются ссылки на один и тот же участок ОП, в который данный сегмент загружается в единственном экземпляре.

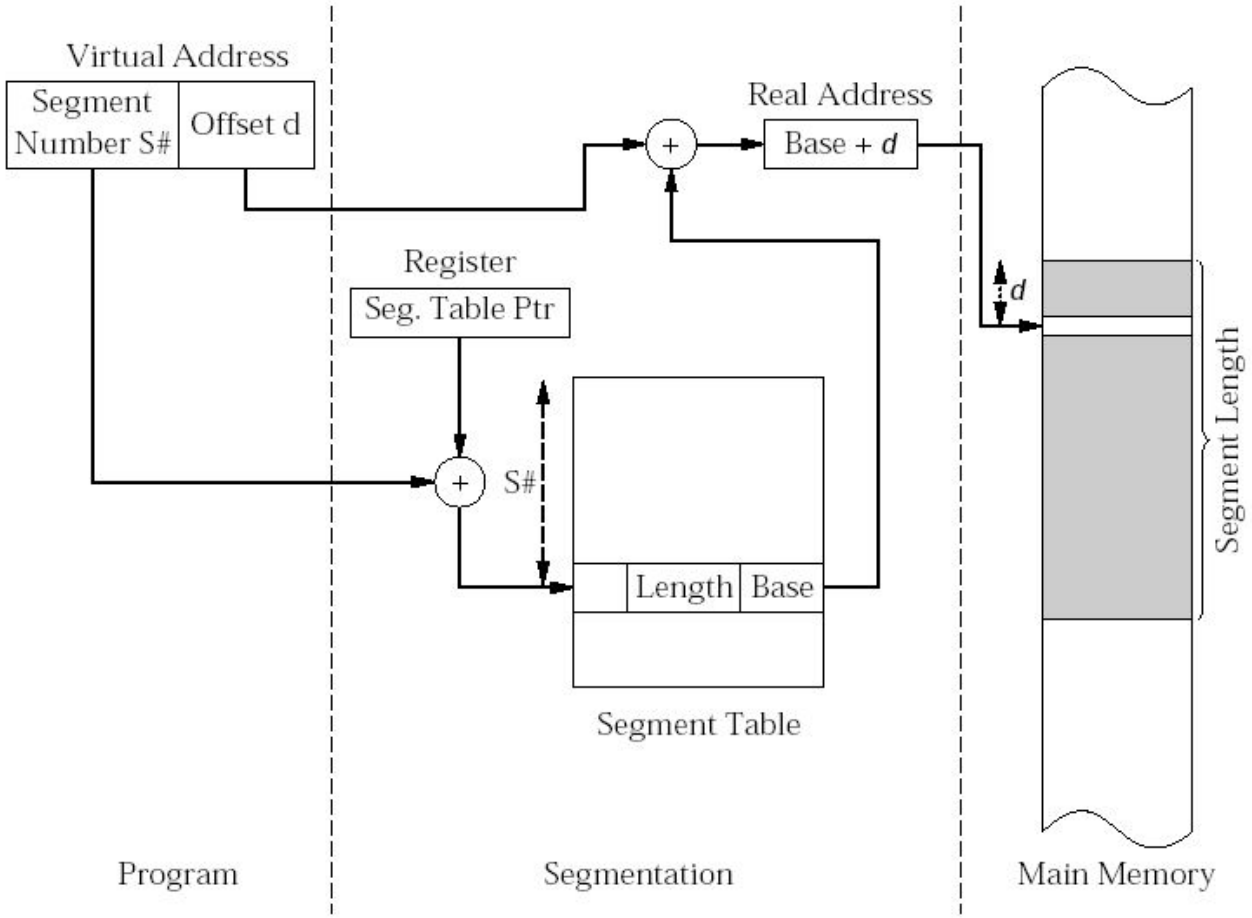
Таблицы сегментов процессов



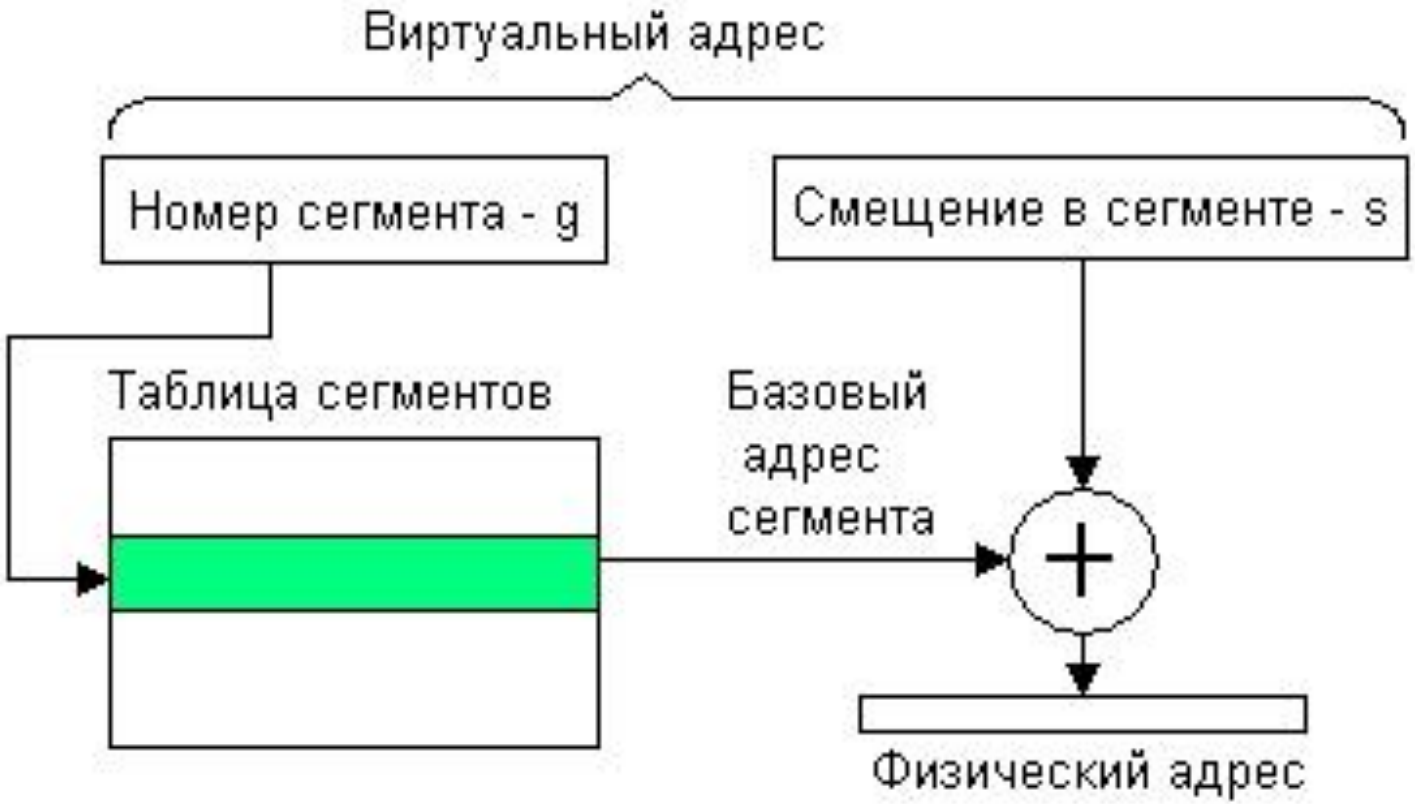
Сегментное распределение: преобразование ВА в ФА

- Система с сегментной организацией функционирует аналогично системе со страничной организацией: время от времени происходят прерывания, связанные с отсутствием нужных сегментов в памяти, при необходимости освобождения памяти некоторые сегменты выгружаются, при каждом обращении к оперативной памяти выполняется преобразование ВА в ФА. Кроме того, при обращении к памяти проверяется, разрешен ли доступ требуемого типа к данному сегменту.
- Виртуальный адрес при сегментной организации памяти может быть представлен парой (*S#, d*), где *S#* - номер сегмента, а *d* - смещение в сегменте. Физический адрес получается путем сложения начального физического адреса сегмента *base*, найденного в таблице сегментов по номеру *S#*, и смещения *d*.

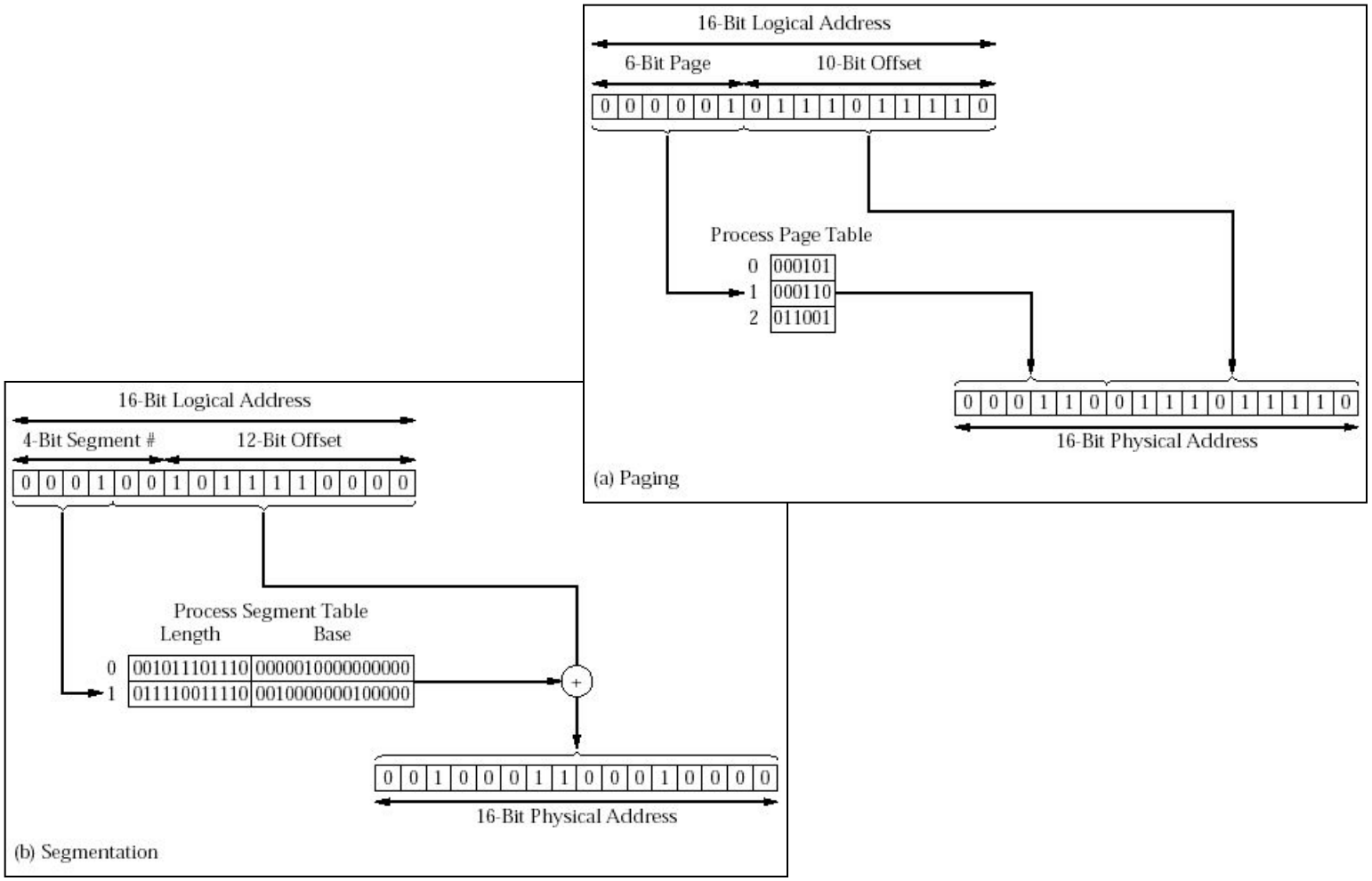
Сегментное распределение: преобразование ВА в ФА



Сегментное распределение: преобразование ВА в ФА



Сравнение страничного и сегментного распределения



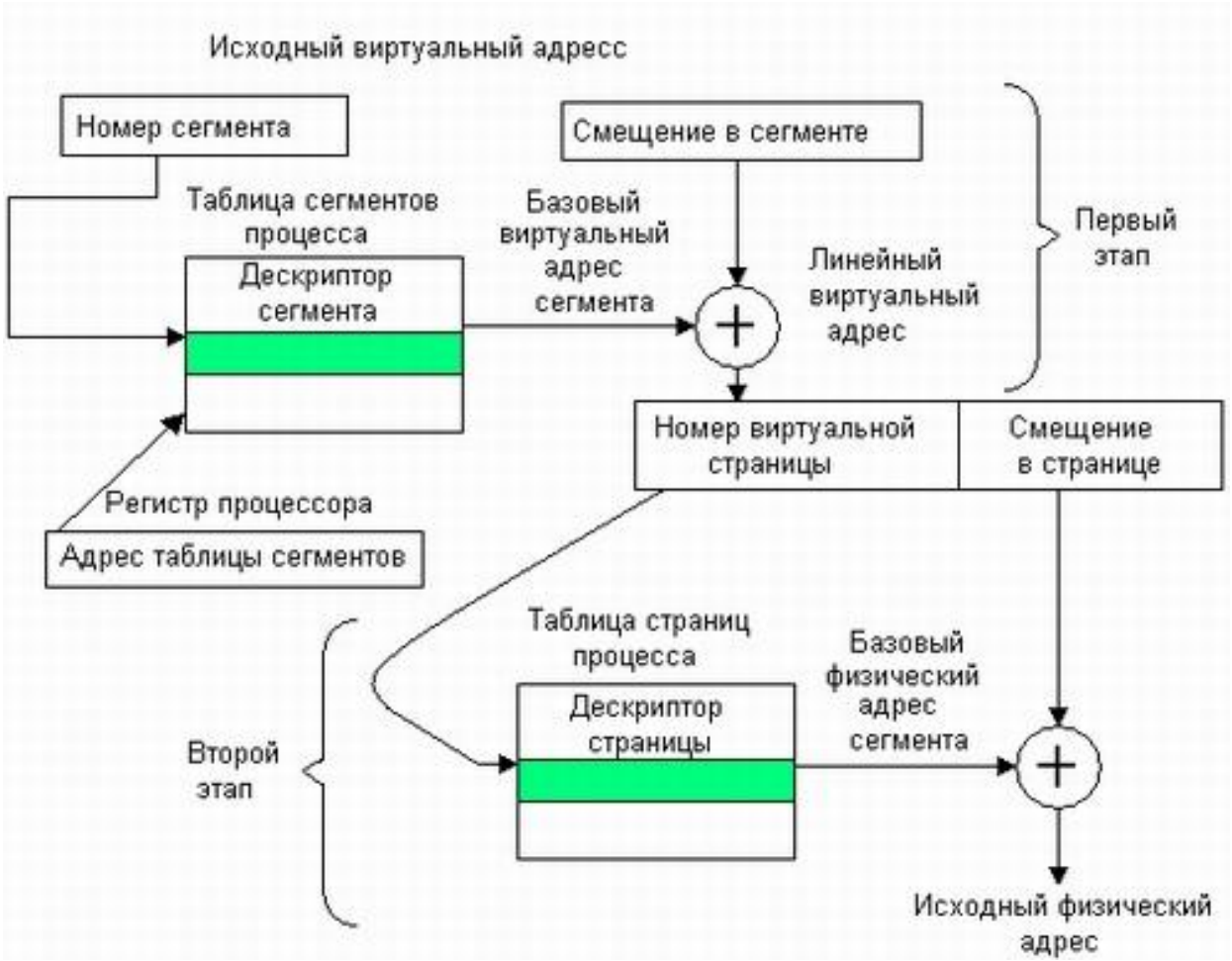
Недостатки сегментного распределения

- Более медленное (по сравнению со страничным распределением) преобразование ВА в ФА в связи с использованием операции сложения.
- Высокий уровень фрагментации оперативной памяти.
- Сложность реализации свопинга, т.к. сегменты разного размера.

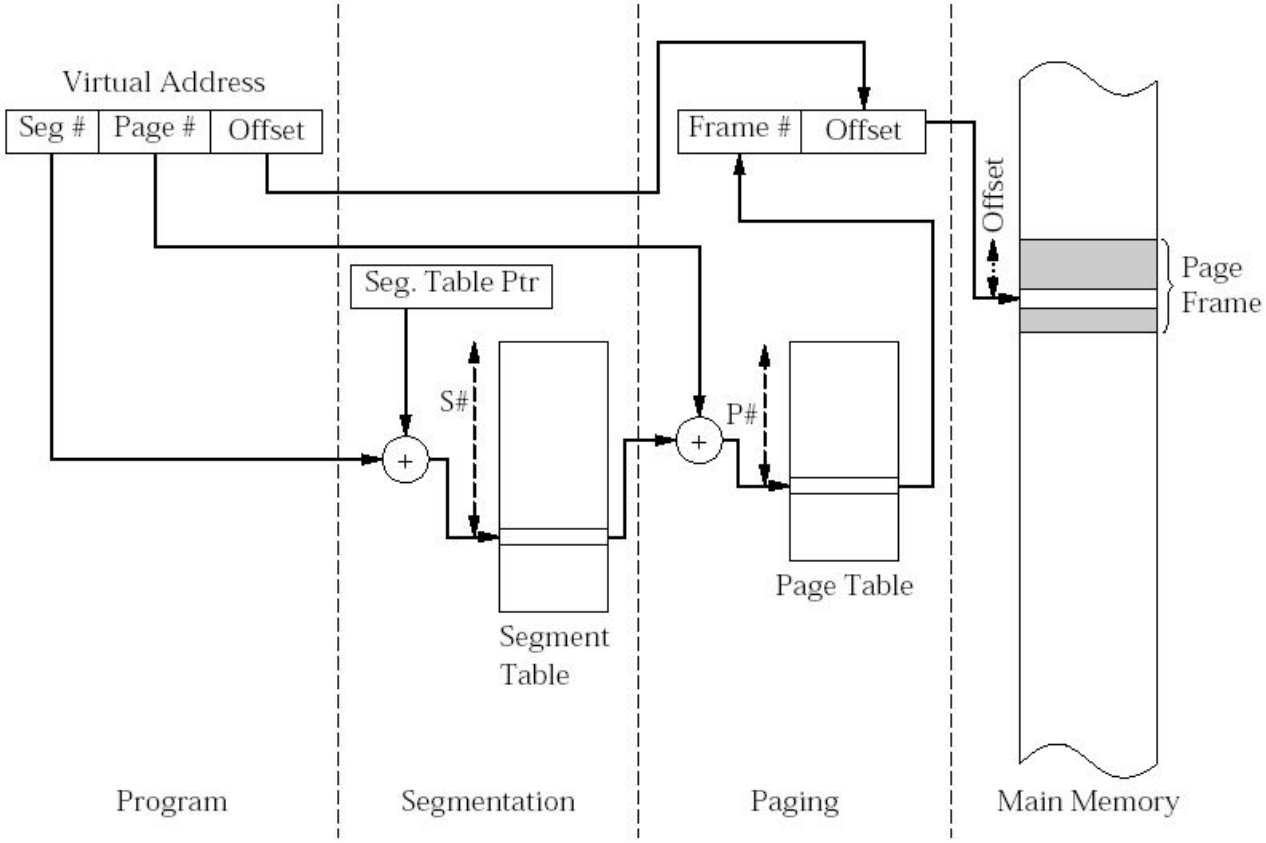
Сегментно-страничное распределение

- Данный метод представляет собой комбинацию страничного и сегментного распределения памяти и, вследствие этого, сочетает в себе достоинства обоих подходов.
- ВАП процесса делится на сегменты, а каждый сегмент в свою очередь делится на виртуальные страницы, которые нумеруются в пределах сегмента.
- Оперативная память делится на физические страницы.
- Загрузка процесса выполняется ОС постранично, при этом часть страниц размещается в оперативной памяти, а часть на диске. Для каждого сегмента создается своя таблица страниц, структура которой полностью совпадает со структурой таблицы страниц, используемой при страничном распределении. Для каждого процесса создается таблица сегментов, в которой указываются адреса таблиц страниц для всех сегментов данного процесса. Адрес таблицы сегментов загружается в специальный регистр процессора, когда активизируется соответствующий процесс.

Сегментно-страничное распределение: преобразование ВА в ФА



Сегментно-страничное распределение: преобразование ВА в ФА



Общие выводы

- страничная виртуальная память организует перемещение данных между памятью и диском страницами – частями ВАП фиксированного и сравнительно небольшого размера (достоинства – высокая скорость обмена, низкий уровень фрагментации; недостатки – сложно организовать защиту данных, разделенных на части механически);
- сегментная виртуальная память предусматривает перемещение данных сегментами – частями ВАП произвольного размера, полученными с учетом смыслового значения данных (достоинства – «осмысленность» сегментов упрощает их защиту; недостатки – медленное преобразование адреса, высокий уровень фрагментации);
- сегментно-страничная виртуальная память сочетает достоинства обоих предыдущих подходов.

Основы управления памятью

Стратегии управления виртуальной памятью (свопинг)

Стратегии управления виртуальной памятью

- *Стратегия выборки (fetch policy)*
- *Стратегия размещения (placement policy)*
- *Стратегия замещения (replacement policy)*

Выборка

- Определяет, в какой момент следует переписать отсутствующую в ОП страницу (сегмент) из внешней памяти в ОП.
- Выборка бывает по запросу и с упреждением.
- В простейшем случае заключается в загрузке страницы (сегмента) с диска в свободную физическую страницу (сегмент) и отображении на эту физическую страницу (сегмент) виртуального адреса, обращение по которому вызвало исключительную ситуацию.

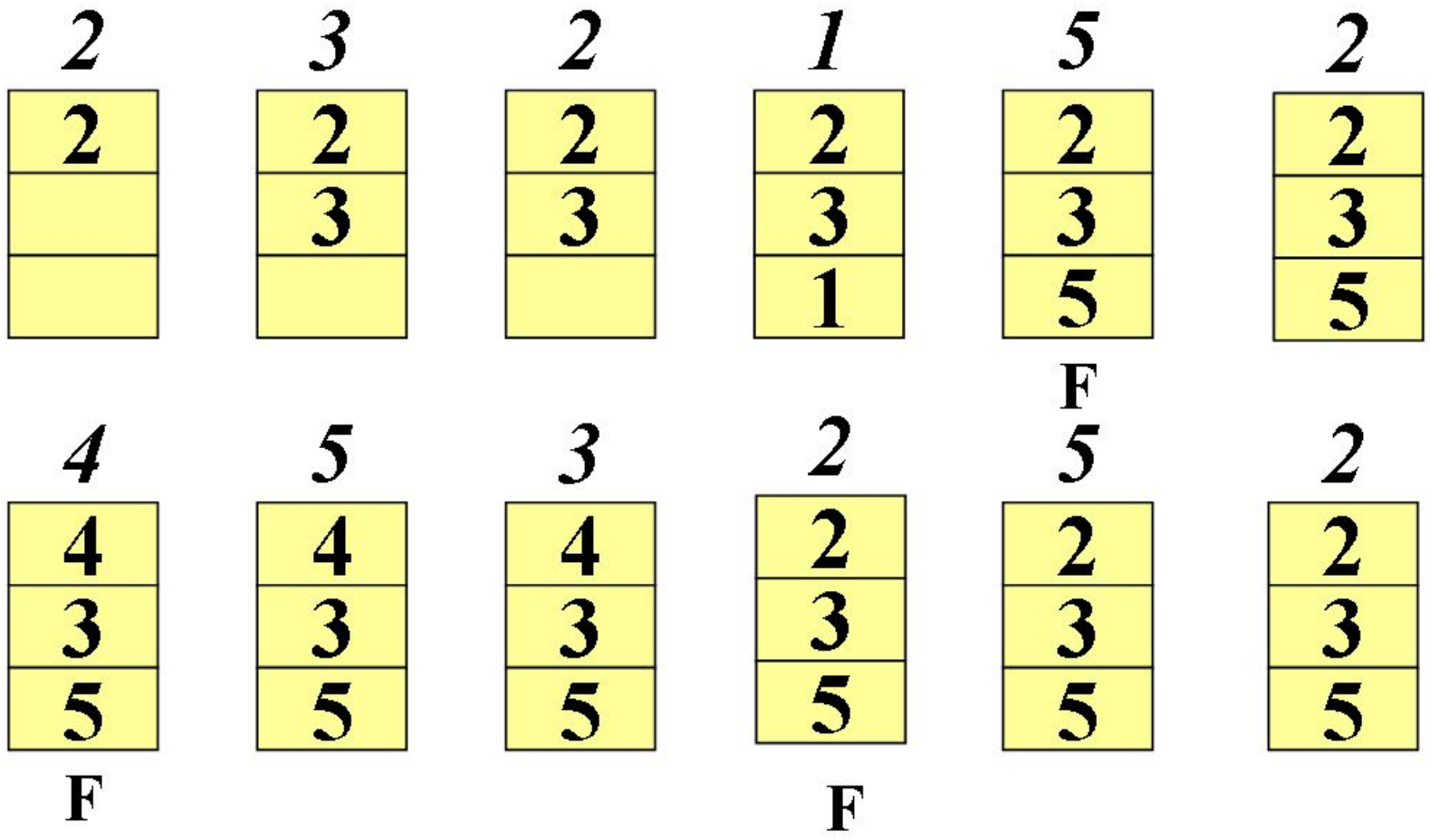
Размещение

- Определяет, в какое место первичной памяти следует поместить поступающую страницу (сегмент).
- В системах со страничной организацией данная стратегия практически не имеет никакого значения, т.к. подходит любой свободный страничный кадр оперативной памяти.
- В системах с сегментной организацией требуется стратегия, аналогичная стратегии с переменными разделами.

Замещение

- Определяет, какую страницу (сегмент) нужно вытолкнуть во внешнюю память, чтобы освободить место.
- Разумная стратегия замещения позволяет оптимизировать хранение в памяти самой необходимой информации.
- Оптимальный алгоритм замещения заключается в том, что бы выгружать ту страницу (сегмент), которая будет запрошена позже всех. Но этот алгоритм не осуществим, т.к. нельзя заранее знать какую страницу (сегмент), когда запросят.

Пример оптимального алгоритма замещения



Пример оптимального алгоритма замещения

- ***1’st page fault*** : страница 1 была вытеснена и заменена страницей 5, т.к. страница 1 в будущем больше не будет вызываться.
- ***2’nd page fault***: страница 2 была вытеснена и заменена страницей 4, т.к. страница 2 будет вызвана позже чем остальные две страницы (страницы 5 и 3).
- ***3’rd page fault***: страница 4 была вытеснена и заменена страницей 2, т.к. страница 4 в будущем больше не будет вызываться.

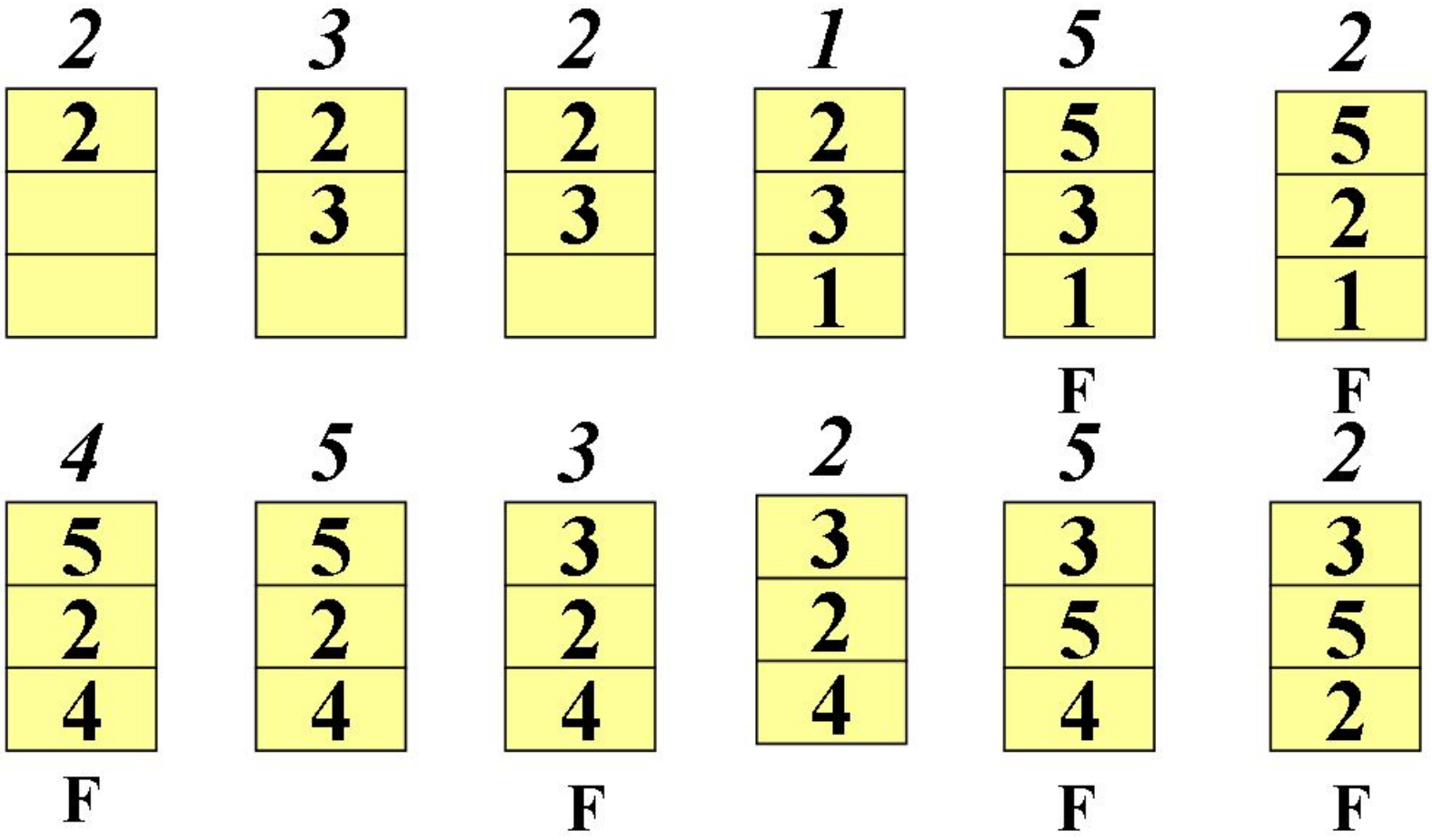
Алгоритмы замещения страниц (свопинга)

- Глобальные – оперируют всей совокупностью страниц оперативной памяти.
- Локальные – оперируют множеством страниц оперативной памяти, принадлежащих конкретному процессу.

Алгоритмы замещения страниц (свопинга)

- **FIFO** (First In First Out) – замещение первой использованной страницы
- **FIFO 2nd Chance** (похож на **clock**)
- **LRU** (Least Recently Used) – замещение дольше всех неиспользовавшихся страниц
- **NRU** (Not Recently Used) или **clock** – замещение не использовавшихся в последнее время страницы
- **NFU** (Not Frequently Used) – замещение наименее часто используемых страниц

Пример действия FIFO



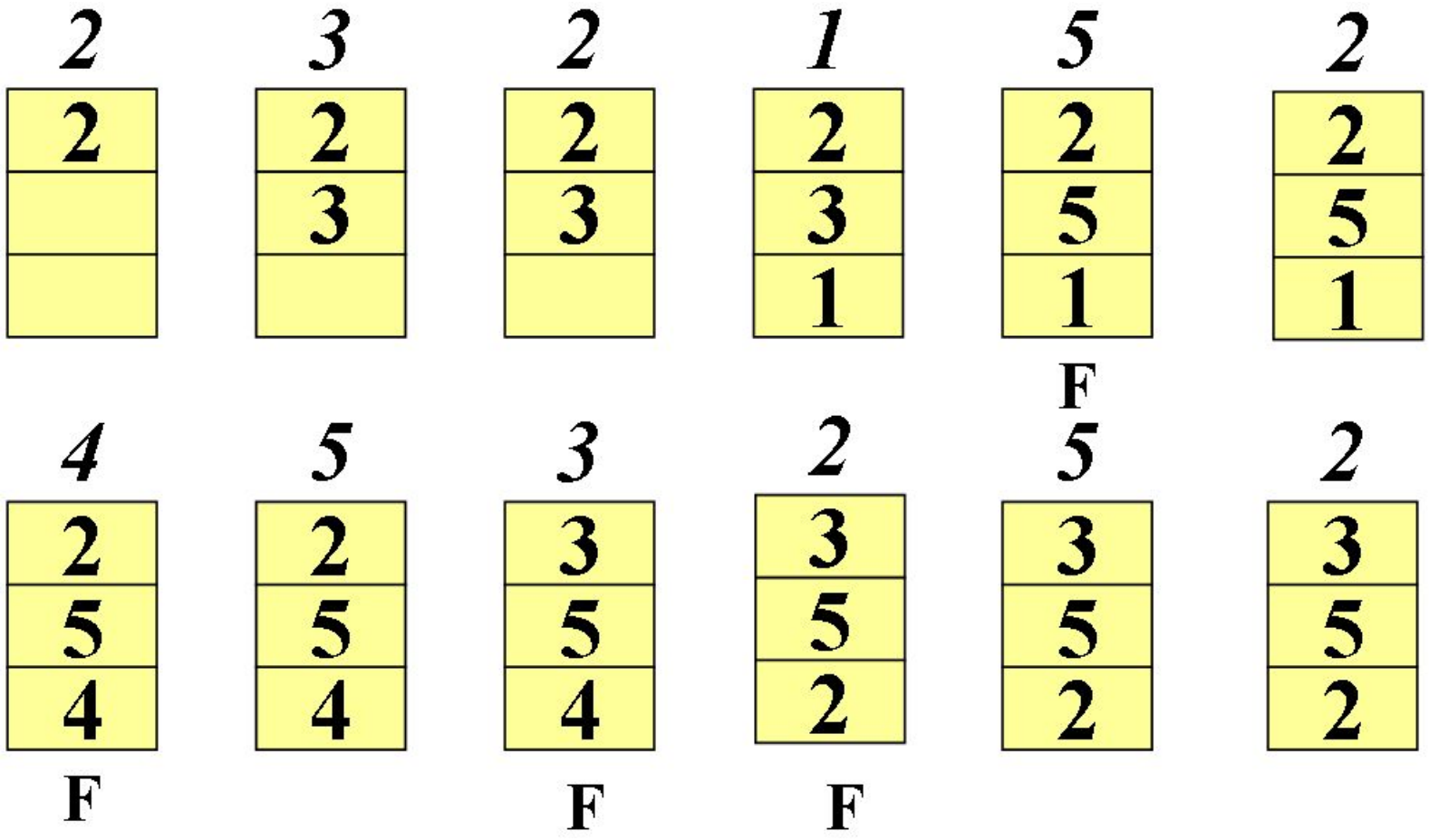
FIFO 2nd Chance

- Модификация алгоритма FIFO, которая использовалась в ранних версиях UNIX.
- Позволяет избежать потери часто используемых страниц с помощью анализа признака использования R для самой «старой» страницы.
- Если признак установлен (R = 1), то страница, в отличие от FIFO, не выталкивается, а очищается бит (R = 0) и страница становится в конец очереди.
- Недостаток – недостаточная эффективность алгоритма, потому что постоянно передвигает страницы по списку. Поэтому лучше хранить описания страничных блоков в виде кольцевого списка и использовать указатель на старейшую страницу – алгоритм NRU (clock).

Алгоритм LRU

- Для замещения выбирается дольше всего неиспользовавшаяся страница.
- Часто используется и считается хорошим.
- Основная проблема – реализация (требуется аппаратная поддержка).

Пример действия LRU



Пример действия LRU

- ***1'st page fault***: page 5 replaces page 3 because page 3 hasn't been referenced in the last two references
- ***2'nd page fault***: page 4 replaces page 1 because page 1 hasn't been referenced in the last two references
- ***3'rd page fault***: page 3 replaces page 2 because page 2 hasn't been referenced in the last two references
- ***4'th page fault***: page 2 replaces page 4 because page 4 hasn't been referenced in the last two references

Реализация LRU №1

- Основана на использовании специального *признака обращения* (reference bit) к странице (требуется аппаратная поддержка).
- Каждой странице назначается свой *счетчик обращений*.
- С некоторым постоянным временным интервалом для каждой страницы выполняется:
  - если признак обращения = 0 (страница не использовалась), увеличить счетчик на 1;
  - если признак обращения = 1 (страница использовалась), обнулить счетчик;
  - сбросить признак обращения.
- Счетчик будет содержать число интервалов, в течение которых страница не использовалась, страница с максимальным значение счетчика – дольше всего не использовавшаяся.

Реализация LRU №2

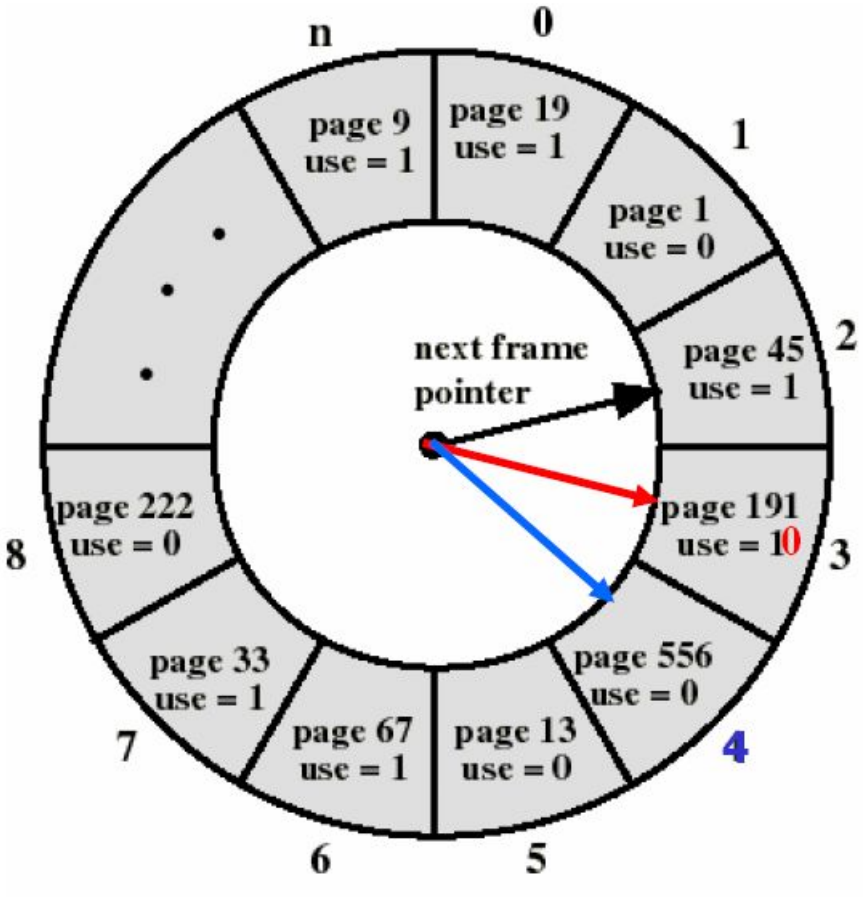
- В некоторых архитектурах (например, Intel) признак обращения отсутствует.
- Для эмуляции признака обращения можно использовать признак достоверности (valid bit), сбрасывая его для возникновения «псевдосбоев» страниц – пример ОС Windows 2000-2008.
- Недостаток – огромное количество дополнительных страничных прерываний.

NRU или clock

- Реализация:
  - все страничные кадры ОП выстраиваются в один большой круг (часы) реализуемый обычным кольцевым списком;
  - “стрелка часов” указывает следующего кандидата на вытеснение движется по списку страниц как стрелка часов;
  - если признак обращения сброшен, значит, страница давно не использовалась, и она – подходящая жертва;
  - если признак обращения установлен, он сбрасывается, и стрелка переводится на следующую страницу.

- Особенности:
  - чем чаще требуются страницы, тем быстрее движется стрелка;
  - при достаточно большом объеме памяти дополнительные расходы невелики;
  - если памяти очень много, точность используемой информации снижается и приходится использовать еще одну стрелку для сброса признаков обращения.

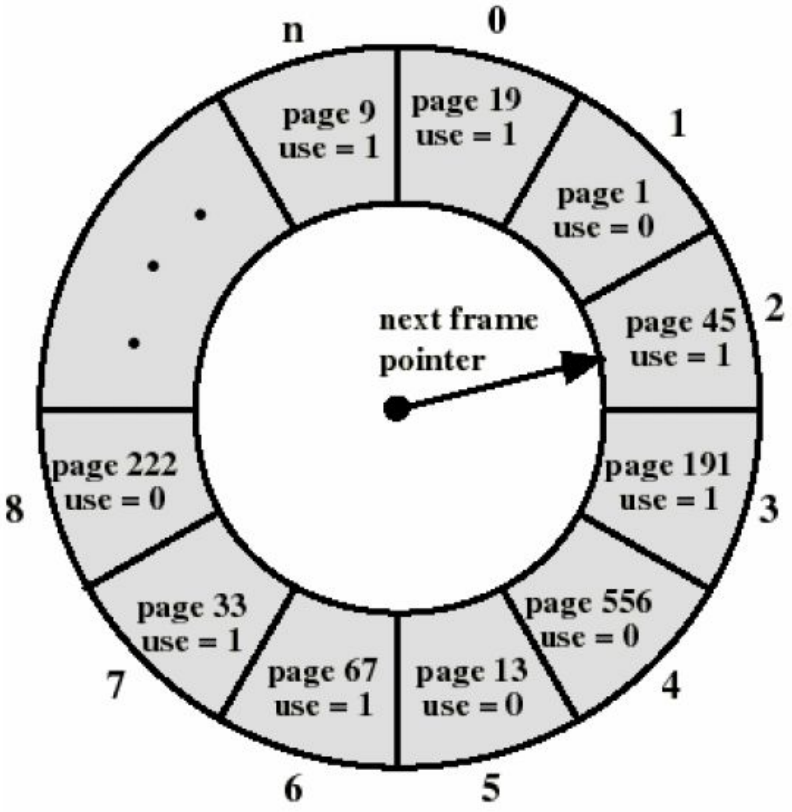
Пример действия NRU



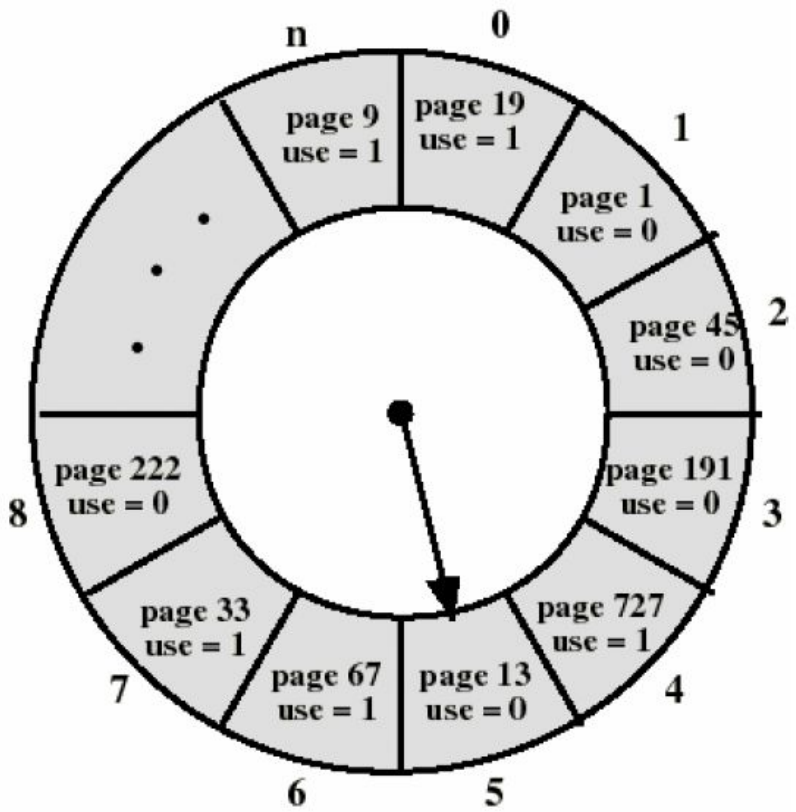
Пример действия NRU

1. A page fault occurs, because page 727 is currently not in main memory.
2. Look for a candidate to be replaced.
3. In a previous replacement step page 45 on page frame 2 had been replaced.
4. Starting from next frame pointer we are looking through the following page frames, resetting their reference bits.
5. Continue this resetting until we find the first page frame with reference bit = 0. Its page frame number is 4, containing page 556 which has to be replaced, because in the past it was not referenced any more: it didn’t use its second chance.

Пример действия NRU



(a) State of buffer just prior to a page replacement



(b) State of buffer just after the next page replacement

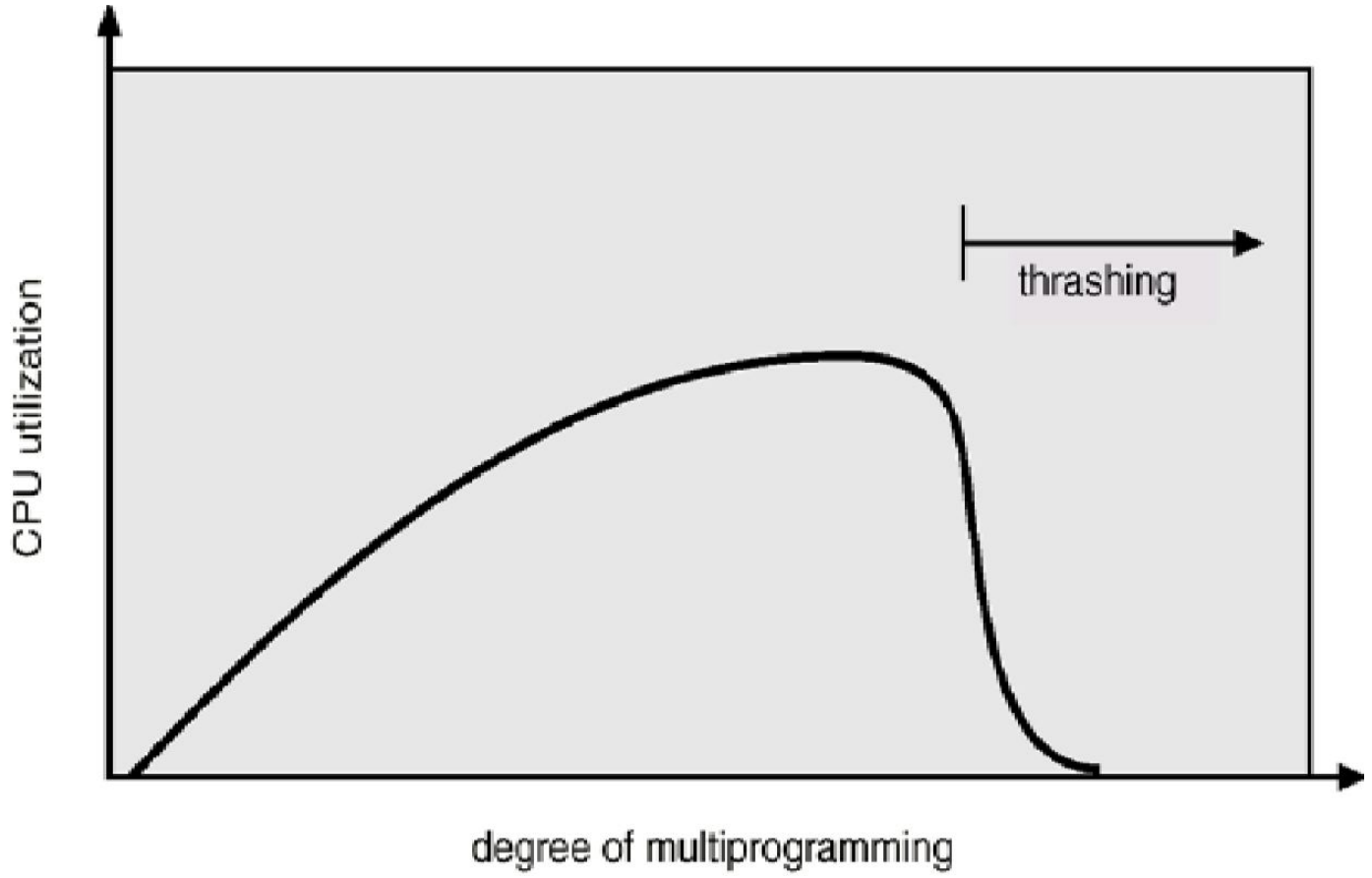
NFU (Not Frequently Used)

- Программная реализация алгоритма, близкого к *LRU*, - алгоритм NFU.
- Для него требуются программные счетчики, по одному на каждую страницу, которые сначала равны нулю. При каждом прерывании по времени ОС сканирует все страницы в памяти и у каждой страницы с установленным флагом обращения увеличивает на единицу значение счетчика, а флаг обращения сбрасывает.
- Таким образом, кандидатом на освобождение оказывается страница с наименьшим значением счетчика, как страница, к которой реже всего обращались.
- Главный недостаток алгоритма NFU состоит в том, что он ничего не забывает. Например, страница, к которой очень часто обращались в течение некоторого времени, а потом обращаться перестали, все равно не будет удалена из памяти, потому что ее счетчик содержит большую величину (глубина истории ограничена разрядностью счетчика).
- Возможна небольшая модификация алгоритма, которая позволяет ему "забывать". Достаточно, чтобы при каждом прерывании по времени содержимое счетчика сдвигалось вправо на 1 бит, а уже затем производилось бы его увеличение для страниц с установленным флагом обращения.
- Другим, уже более устойчивым недостатком алгоритма является длительность процесса сканирования таблиц страниц.

Понятие «trashing»

- Высокая частота страничных прерываний называется *трешинг* (*thrashing*).
- Процесс находится в состоянии трешинга, если он занимается подкачкой страниц больше времени, чем выполнением.
- Критическая ситуация такого рода возникает вне зависимости от конкретных алгоритмов замещения.
- В результате все процессы попадают в очередь запросов на свопинг, а очередь процессов в состоянии готовности пустеет.
- ОС видит это и постепенно увеличивает степень мультипрограммирования.
- Таким образом, пропускная способность системы падает из-за трешинга.

Понятие «trashing»



Решение проблемы trashing

- Эффект трешинга, возникающий при использовании глобальных алгоритмов, может быть ограничен за счет использования локальных алгоритмов замещения.
- В этом случае если даже один из процессов попадает в трешинг, это напрямую не сказывается на других процессах.
- Однако этот процесс много времени проводит в очереди на свопинг своих страниц, затрудняя подкачку страниц остальных процессов.