Управление памятью

Виртуальная память

Виртуальным называется ресурс, который пользователю или пользовательской программе представляется обладающим свойствами, которыми он в действительности не обладает. Так, например, пользователю может быть предоставлена виртуальная оперативная память, размер которой превосходит всю имеющуюся в системе реальную оперативную память. Пользователь пишет программы так, как будто в его распоряжении имеется однородная оперативная память большого объема, но в действительности все данные, используемые программой, хранятся на одном или нескольких разнородных запоминающих устройствах, обычно на дисках, и при необходимости частями отображаются в реальную память.

Таким образом, **виртуальная память** - это совокупность программно-аппаратных средств, позволяющих пользователям писать программы, размер которых превосходит имеющуюся оперативную память; для этого виртуальная память 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;
- преобразует виртуальные адреса в физические.

Все эти действия выполняются *автоматически*, без участия программиста, то есть механизм виртуальной памяти является прозрачным по отношению к пользователю.

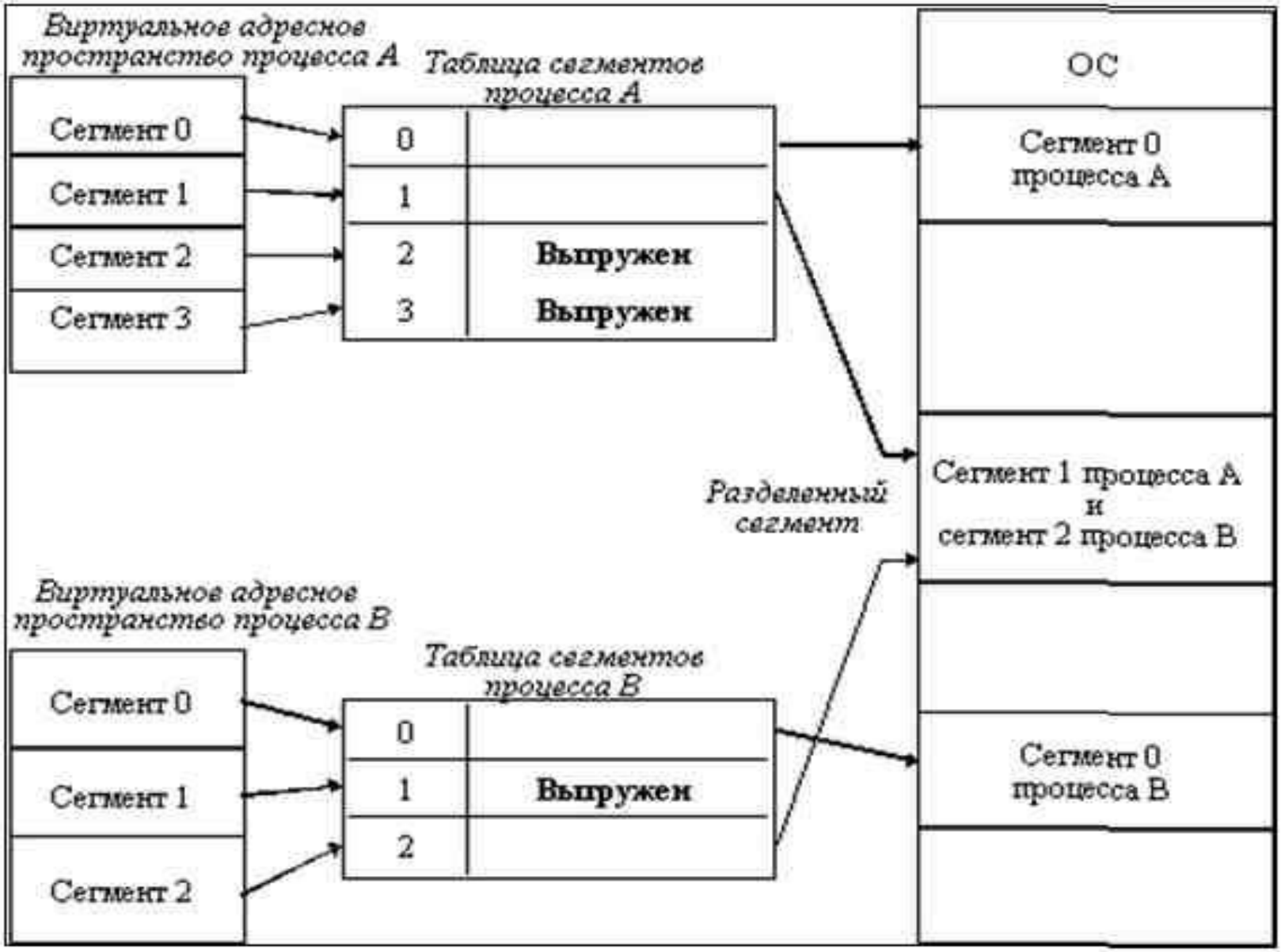
Наиболее распространенными реализациями виртуальной памяти является страничное, сегментное и странично-сегментное распределение памяти, а также свопинг.

Виртуализация памяти в ранних операционных системах осуществлялась на основе свопинга. **Свопинг** (swapping) - способ управления памятью, при котором образы процессов выгружаются на диск и возвращаются в оперативную память целиком. Свопинг представляет собой частный, более простой в реализации, вариант виртуальной памяти, позволяющий совместно использовать оперативную память и диск.

Сегментная организация памяти

Виртуальное адресное пространство 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.

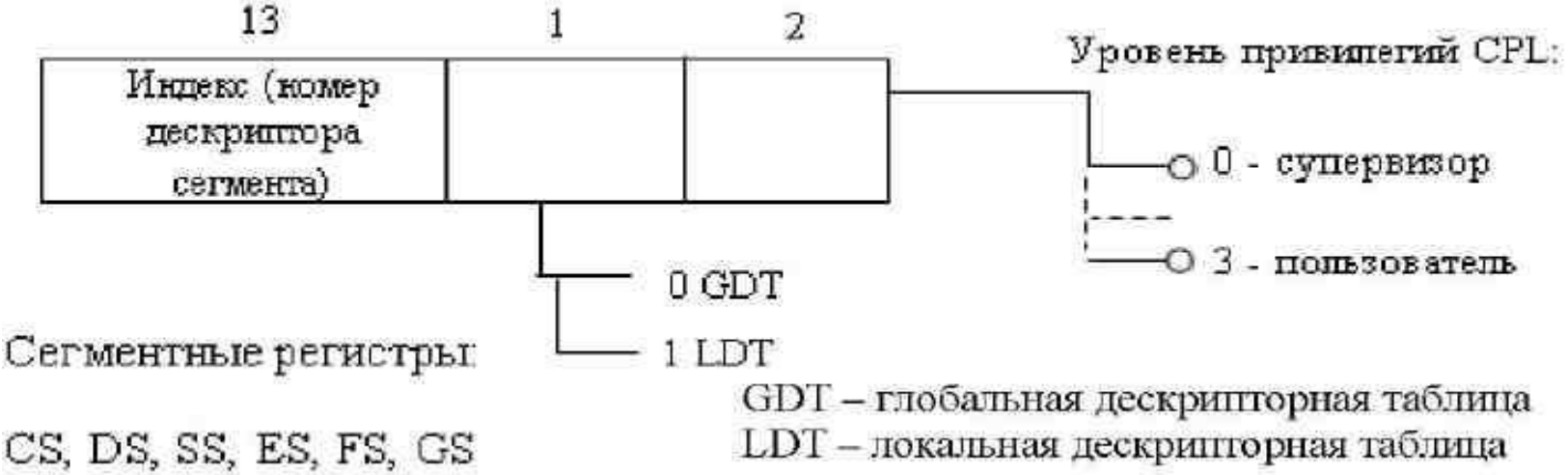
При загрузке процесса часть сегментов помещается в оперативную память (при этом для каждого из этих сегментов операционная система подыскивает подходящий участок свободной памяти), а часть сегментов размещается в дисковой памяти. Сегменты одной программы могут занимать в оперативной памяти несмежные участки. Во время загрузки система создает таблицу сегментов процесса, 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 виртуальные адресные пространства нескольких процессов включают один и тот же сегмент, то в таблицах сегментов этих процессов делаются ссылки на один и тот же участок оперативной памяти, в который данный сегмент загружается в единственном экземпляре.



Распределение памяти сегментами

Сегментная организация на примере Intel

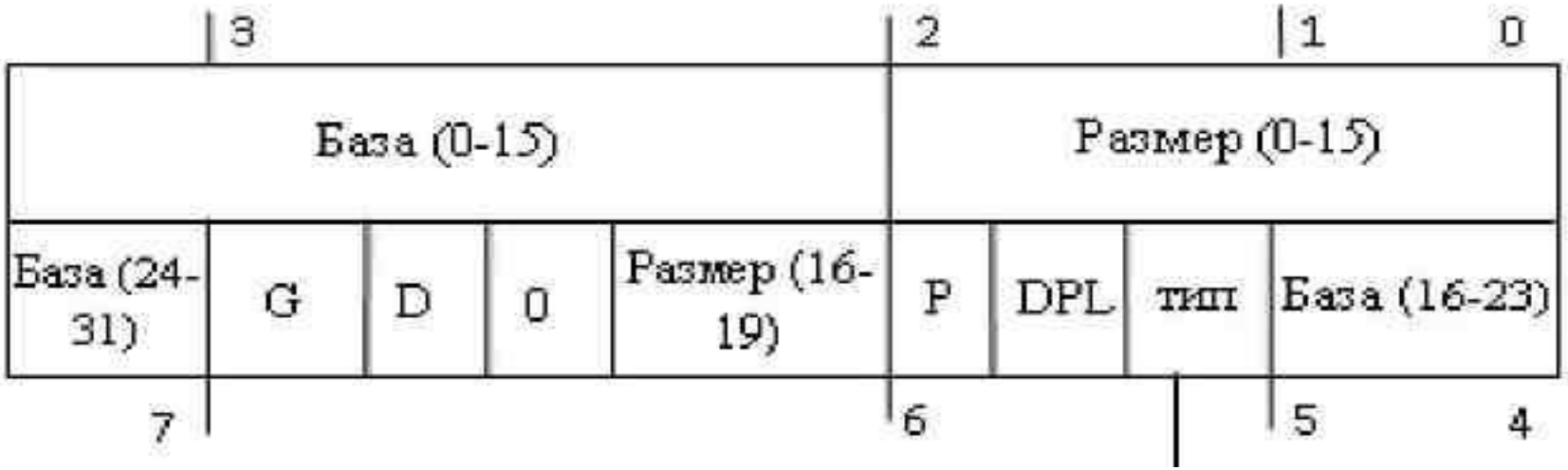
Сегментную организацию памяти рассмотрим на примере процессора Intel. В сегментном механизме управления памятью используются следующие структуры данных. Селектор -это структура данных, которая служит для выбора записи в глобальной или локальной дескрипторной таблице. Селекторы хранятся в сегментных регистрах.

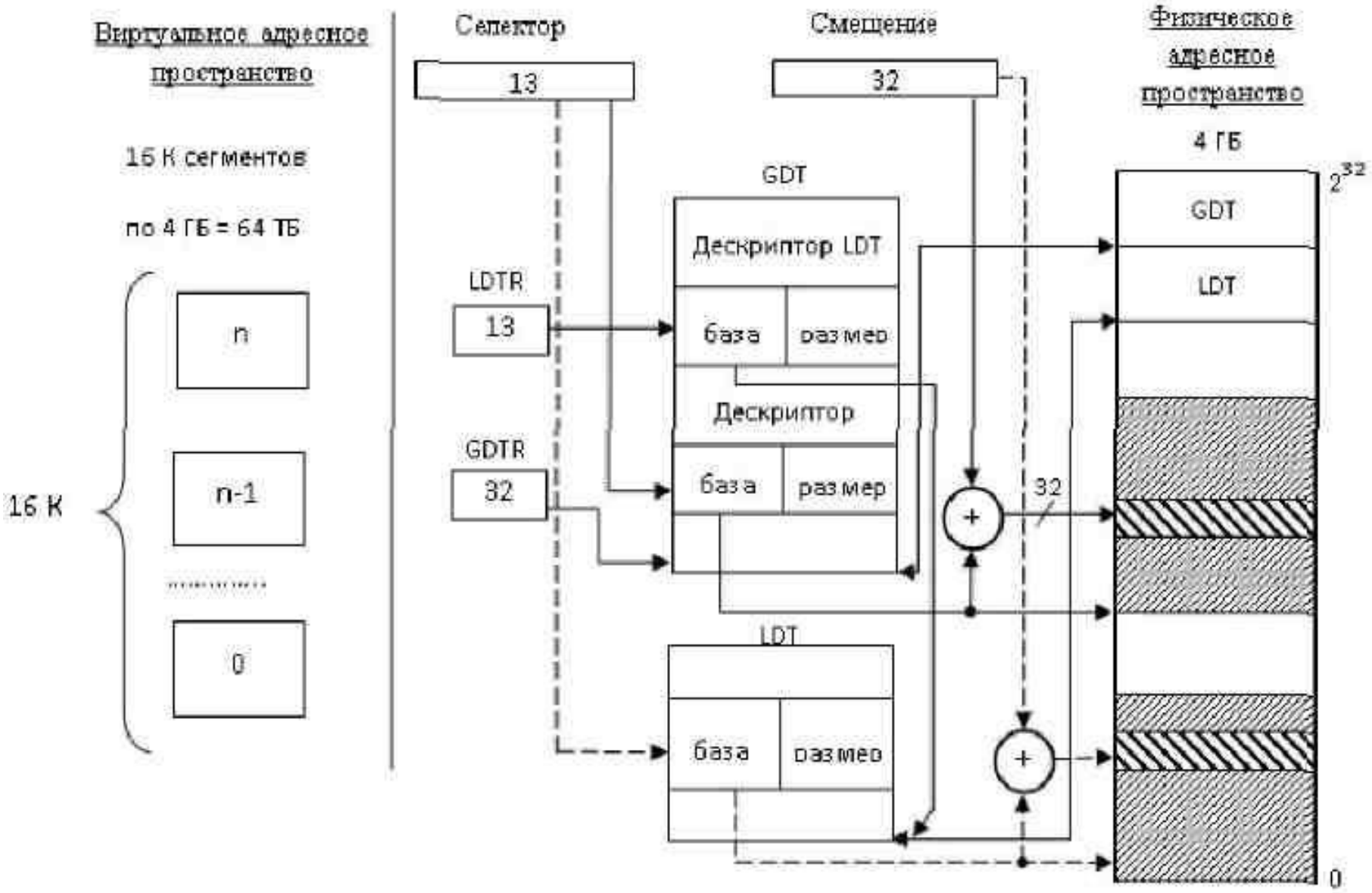


Регистр GDTR описывает положение глобальной дескрипторной таблицы в физической памяти.

Формат дескриптора сегмента данных и кода показан на рисунке. Размер дескриптора 8 байт. Бит G определяет способ изменения размера сегмента: в байтах (режим VM) или в страницах. Бит D определяет выравнивание сегментов по 32-битной или 16-битной границе. Бит P - бит присутствия сегмента в физической памяти. Если он не установлен, то сегмент выгружен на диск. Обращение к сегменту невозможно и приводит к прерыванию.

Биты DPL определяют уровень привилегий дескриптора, то есть возможность обращения к сегменту с данным уровнем привилегий задачи.





Детализация схемы преобразования адресов

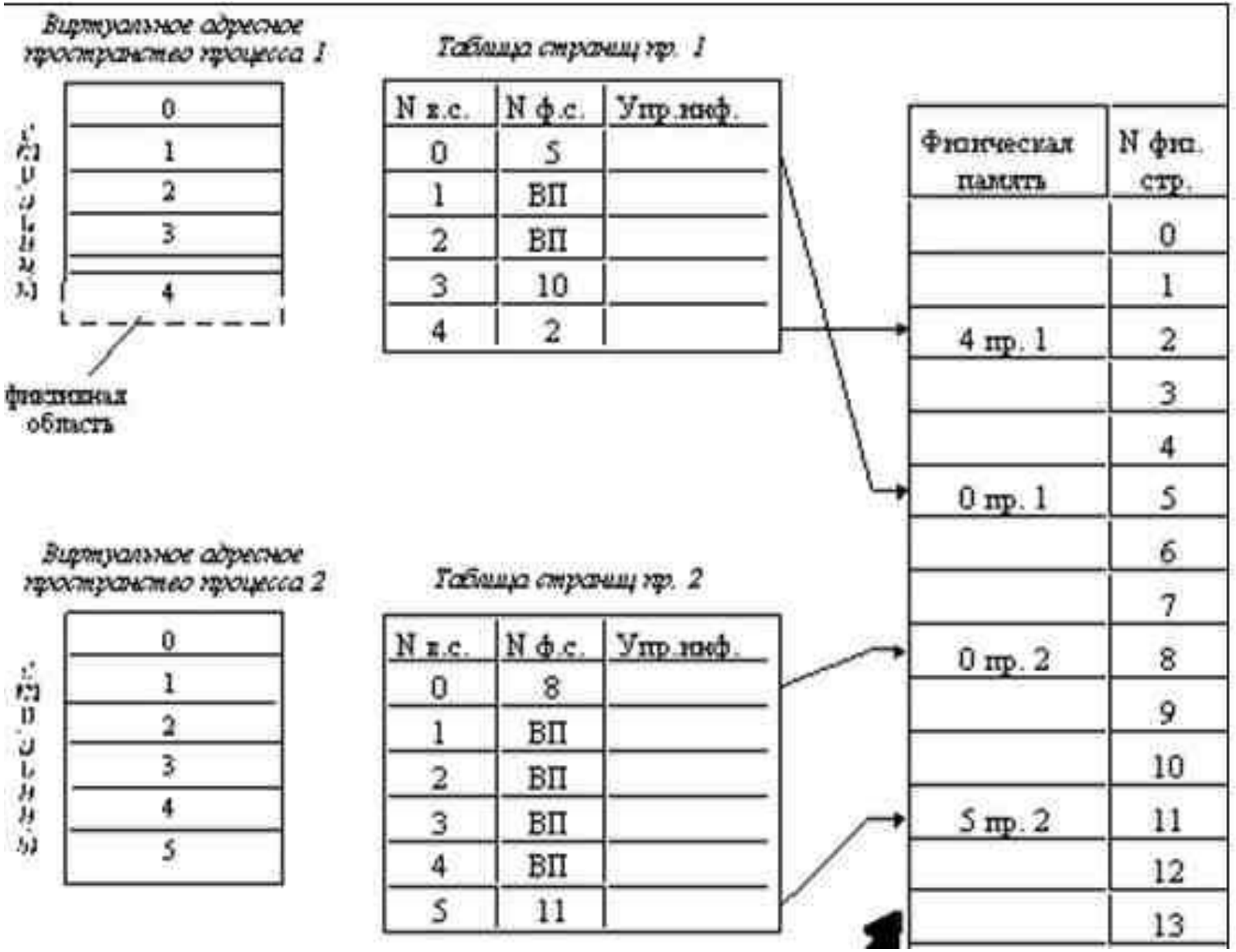
Страничная организация памяти

Виртуальное адресное пространство каждого процесса делится на части одинакового, фиксированного для данной системы размера, называемые виртуальными страницами. В общем случае размер виртуального адресного пространства не является кратным размеру страницы, поэтому последняя страница каждого процесса дополняется фиктивной областью.

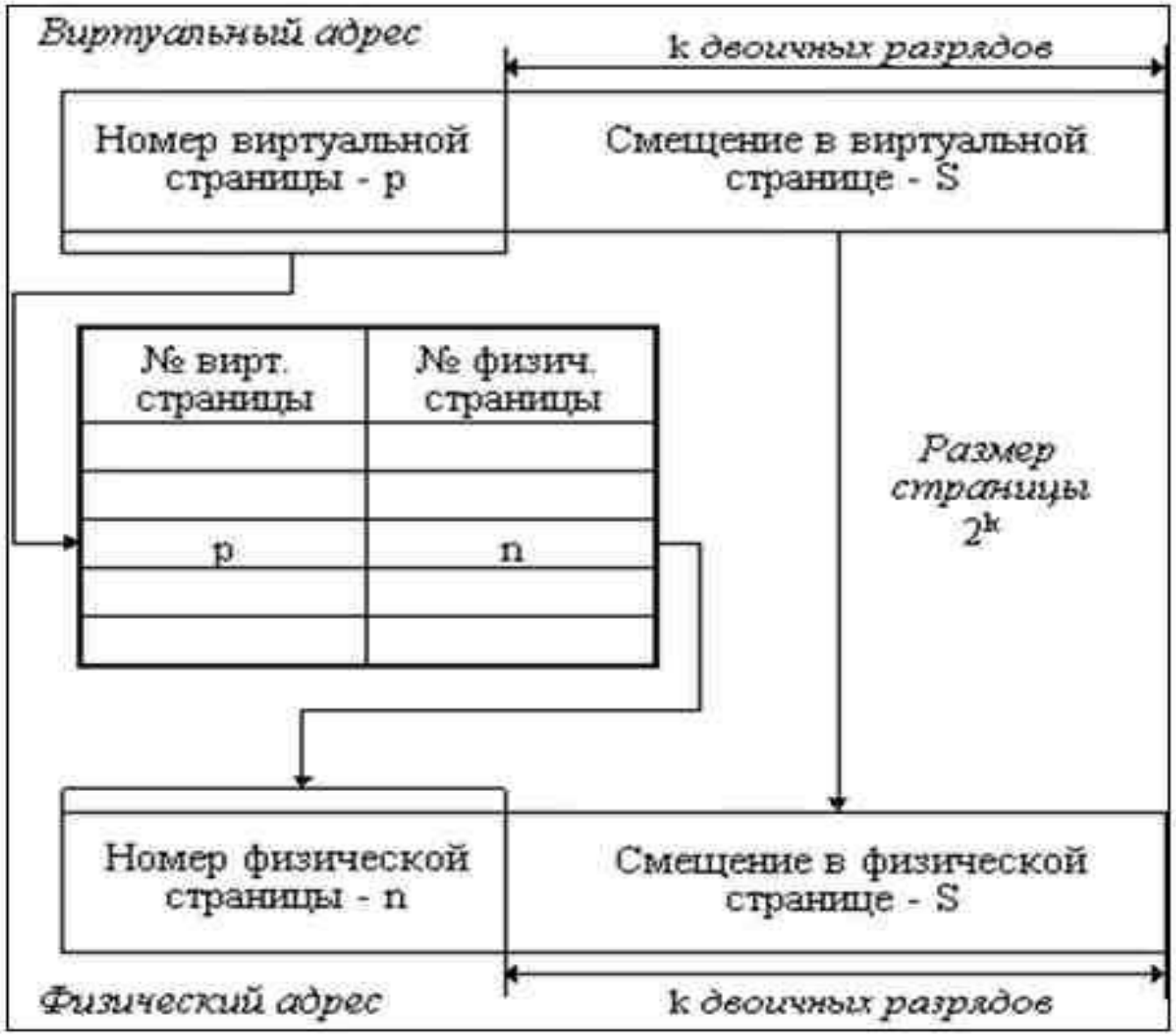
Вся оперативная память машины также делится на части такого же размера, называемые физическими страницами (или блоками).

Размер страницы обычно выбирается равным степени двойки: 512, 1024 и т.д., это позволяет упростить механизм преобразования адресов.

При загрузке процесса часть его виртуальных страниц помещается в оперативную память, а остальные - на диск. Смежные виртуальные страницы не обязательно располагаются в смежных физических страницах. При загрузке операционная система создает для каждого процесса информационную структуру - таблицу страниц, в которой устанавливается соответствие между номерами виртуальных и физических страниц для страниц, загруженных в оперативную память, или делается отметка о том, что виртуальная страница выгружена на диск. Кроме того, в таблице страниц содержится управляющая информация, такая как признак модификации страницы, признак невыгружаемости, признак обращения к странице (используется для подсчета числа обращений за определенный период времени) и другие данные, формируемые и используемые механизмом виртуальной памяти.



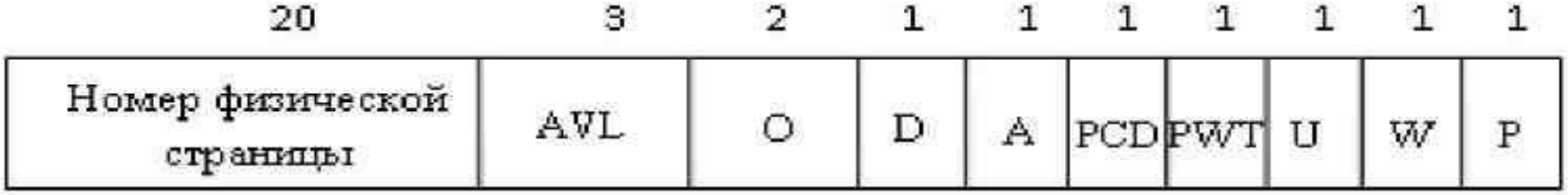
Страничное распределение памяти (ВП – внешняя память)



Преобразование виртуального адреса в физический

Страничная организация на примере Intel

Структура данных, используемая для страничного механизма управления памятью - это дескриптор страниц



Общее число страниц, адресуемых через дескриптор страниц, составляет 2^20. Часть линейного адреса, отводимая под смещение внутри страницы, таким образом, составляет 32-20 = 12 бит. Размер страницы составляет 2^12 = 4 К.

Поля в структуре дескриптора станиц имеют следующее назначение: AVL - резерв ОС; D - признак модификации; A - признак того, был ли доступ; PCD, PWT - биты управления кэшированием страницы; U - пользователь/супервизор; W - разрешение записи в страницу; P - бит присутствия страницы в физической памяти.

Хранение таблицы страниц целиком привело бы к нерациональному использованию физической памяти. Поэтому в архитектуре i386 используется двухуровневый механизм преобразования линейного виртуального адреса в физический.

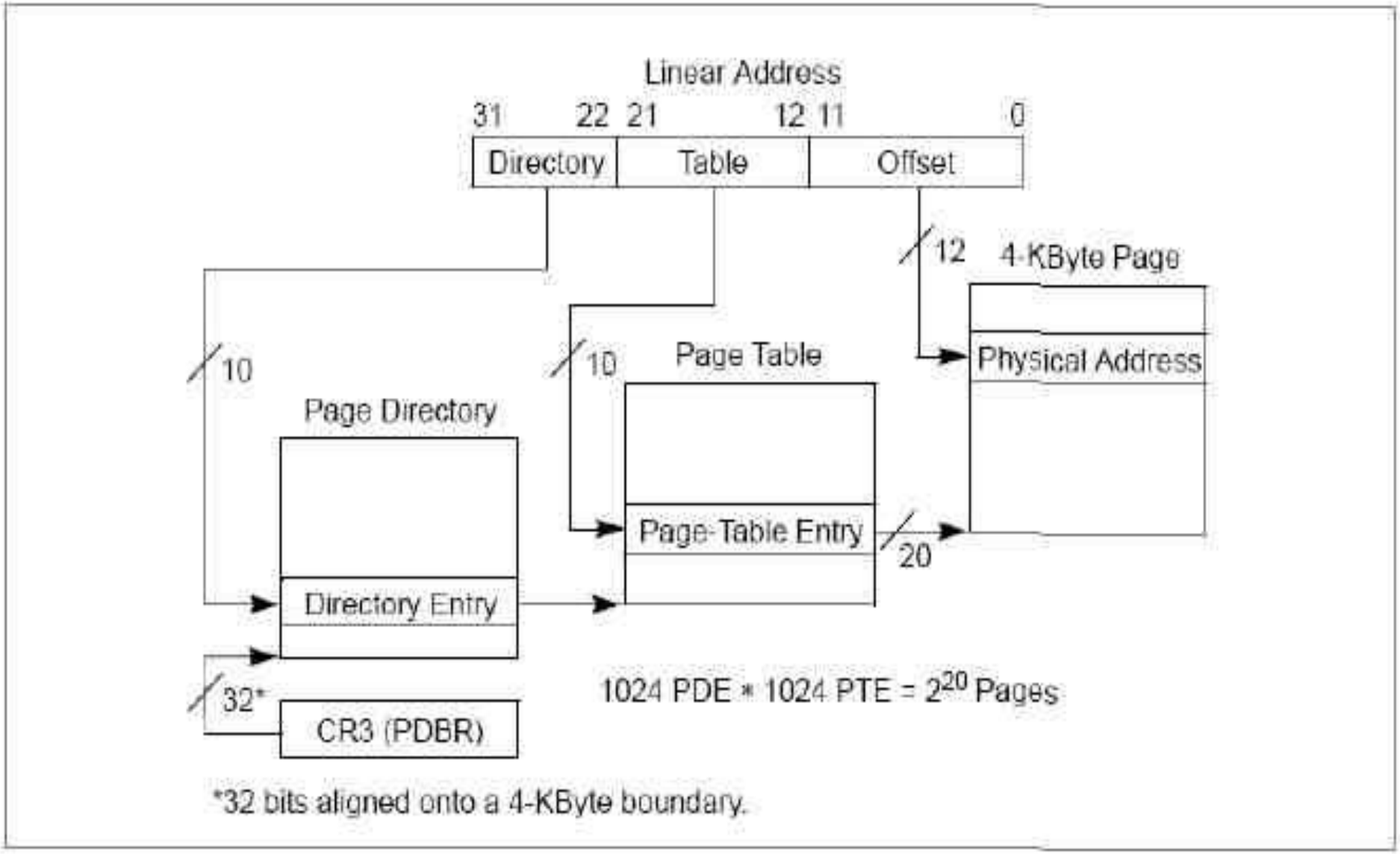
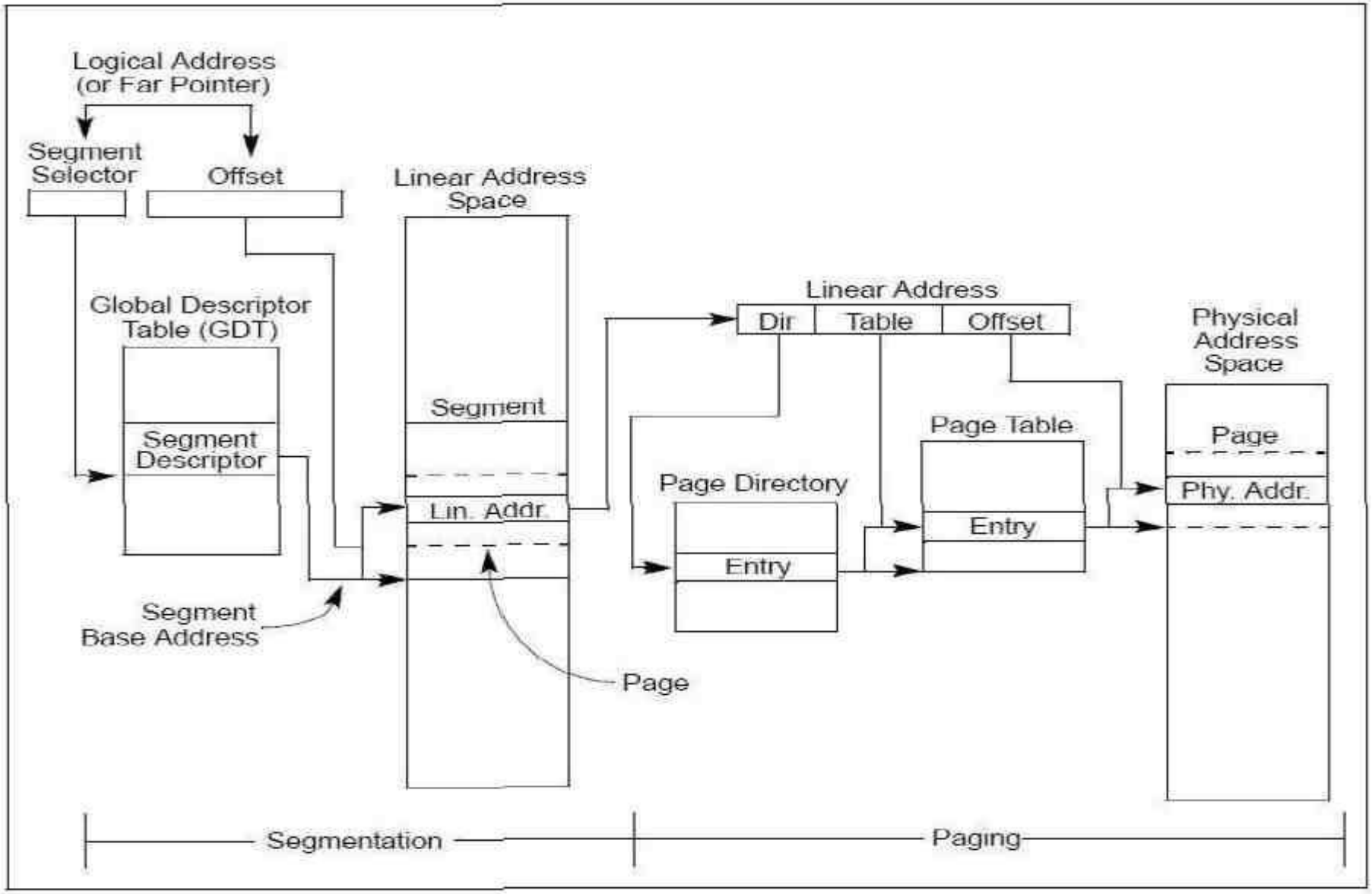


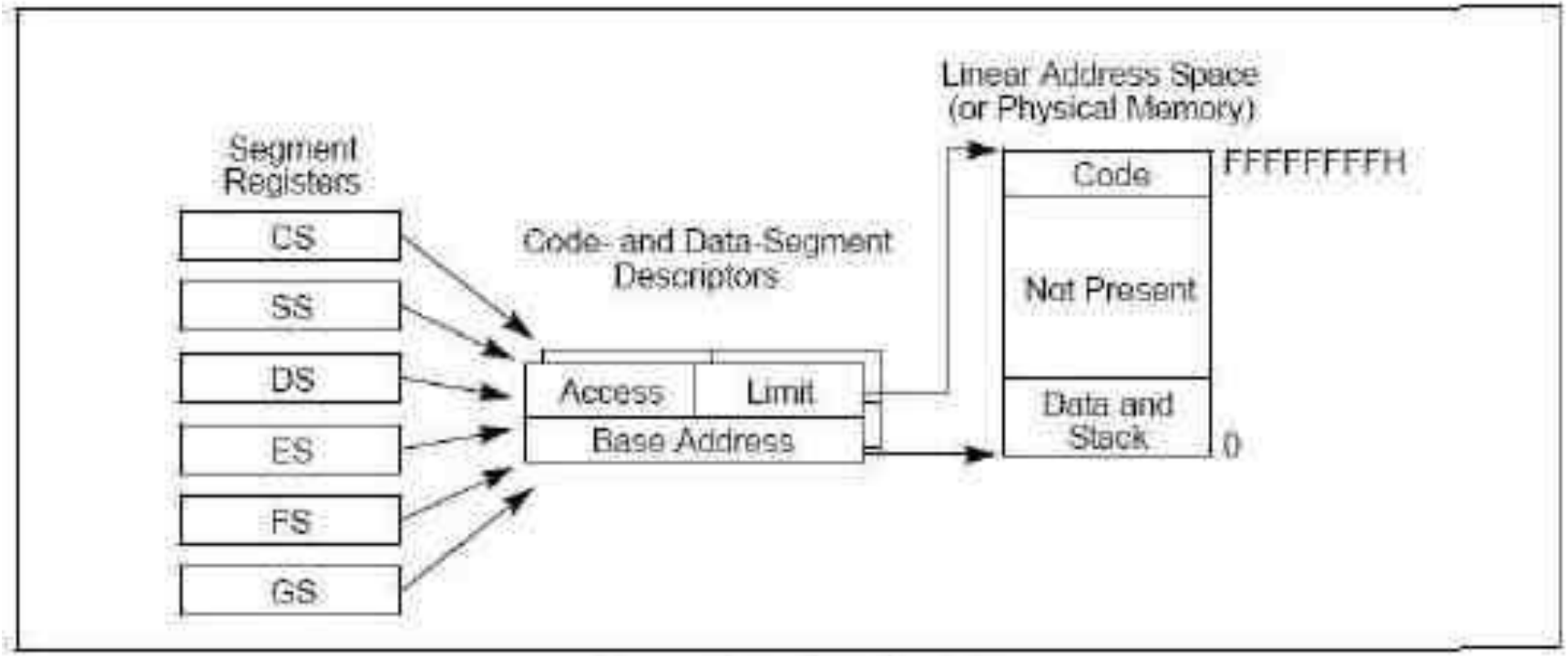
Схема преобразования линейного адреса в физический

Сегментно-страничная организация памяти



Сегментно-страничная организация памяти на примере Intel

Необходимость поддержки сегментации в значительной мере есть данью традиции и связана с поддержкой обратной совместимости. Современные ОС часто обходят такую сегментную организацию, используя в системе только несколько общих сегментов. При этом каждый из них задают селектором, в дескрипторе которого поле **база** (base) равно 0, а поле **размер** (limit) – максимальному адресу линейной памяти. Сдвиг логического адреса всегда будет равен линейному адресу, а следовательно линейный адрес можно формировать в программе, фактически переходя к чисто страничной организации.



Плоская модель памяти

Алгоритмы замещения страниц

Когда сама страница отсутствует в памяти, происходит ошибка отсутствия страницы . Для получения страницы требуется обращение к диску, занимающее в зависимости от используемого диска несколько миллисекунд.

При возникновении ошибки отсутствия страницы операционная система должна выбрать выселяемую (удаляемую из памяти) страницу, чтобы освободить место для загружаемой страницы. Если предназначенная для удаления страница за время своего нахождения в памяти претерпела изменения, она должна быть переписана на диске, чтобы привести дисковую копию в актуальное состояние. Но если страница не изменялась (например, она содержала текст программы), дисковая копия не утратила своей актуальности и перезапись не требуется. Тогда считываемая страница просто пишется поверх выселяемой.

Следует заметить, что проблема замещения страниц имеет место и в других областях информационных технологий. Это в частности обслуживание кэш-памяти компьютера и ведение кэша часто используемых веб-страниц на веб-сервере.

Рассмотрим наиболее часто используемые алгоритмы замещения страниц.

Алгоритм исключения недавно использовавшейся страницы

Чтобы позволить операционной системе осуществить сбор статистики востребованности страниц, большинство компьютеров, использующих виртуальную память, имеют два бита состояния, *R* и *M*, связанных с каждой страницей. Бит *R* устанавливается при каждом обращении к странице (при чтении или записи). Бит *M* устанавливается, когда в страницу ведется запись (то есть когда она модифицируется). Эти биты присутствуют в каждой записи таблицы страниц. Указанные биты должны обновляться при каждом обращении к памяти, поэтому необходимо, чтобы их значения устанавливались аппаратной частью. После установки бита в 1 бит сохраняет это значение до тех пор, пока не будет сброшен операционной системой.

При запуске процесса все записи в его таблице страниц помечаются отсутствующими в памяти. Как только произойдет обращение к странице, возникнет ошибка отсутствия страницы. Тогда операционная система устанавливает бит *R* и режим доступа к странице READ ONLY. Если впоследствии страница модифицируется, возникает другая ошибка страницы, позволяющая операционной системе установить бит *M* и изменить режим доступа к странице на чтение-запись (READ/WRITE).

Биты *R* и *M* могут использоваться для создания следующего простого алгоритма замещения страниц. При запуске процесса оба страничных бита для всех его страниц устанавливаются операционной системой в 0. Время от времени бит *R* сбрасывается, чтобы отличить те страницы, к которым в последнее время не было обращений, от тех, к которым такие обращения были.

При возникновении ошибки отсутствия страницы операционная система просматривает все страницы и на основе текущих значений принадлежащих им битов *R* и *M* делит их на четыре категории:

- Класс 0: в последнее время не было ни обращений, ни модификаций.
- Класс 1: обращений в последнее время не было, но страница модифицирована.
- Класс 2: в последнее время были обращения, но модификаций не было.
- Класс 3: в последнее время были и обращения, и модификации.

Алгоритм исключения недавно использовавшейся страницы (Not Recently Used -**NRU**) удаляет произвольную страницу, относящуюся к самому низкому непустому классу.

Алгоритм «первым пришел, первым ушел»

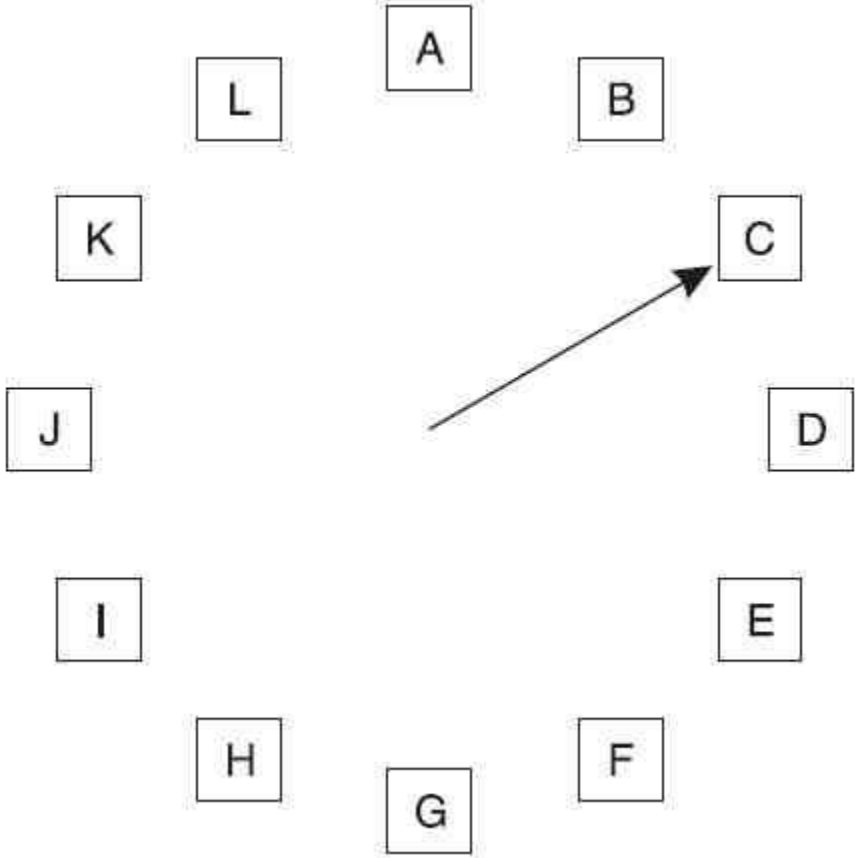
Другим низкозатратным алгоритмом замещения страниц является алгоритм FIFO (First In, First Out — первым пришел, первым ушел). Операционная система ведет список всех страниц, находящихся на данный момент в памяти, причем совсем недавно поступившие находятся в хвосте, поступившие раньше всех — в голове списка. При возникновении ошибки отсутствия страницы удаляется страница, находящаяся в голове списка, а к его хвосту добавляется новая страница. Однако а реальности самая старая страница все еще может пригодиться. Поэтому принцип FIFO в чистом виде используется редко.

Алгоритм «второй шанс»

Простой модификацией алгоритма FIFO, исключающей проблему удаления часто востребуемой страницы, может стать проверка бита *R* самой старой страницы. Если его значение равно нулю, значит, страница не только старая, но и невостребованная, поэтому она тут же удаляется. Если бит *R* имеет значение 1, он сбрасывается, а страница помещается в конец списка страниц и время ее загрузки обновляется, как будто она только что поступила в память. Затем поиск продолжается.

Алгоритм «часы»

При всей своей логичности алгоритм «второй шанс» слишком неэффективен, поскольку он постоянно перемещает страницы в своем списке. Лучше содержать все страничные блоки в циклическом списке в виде часов. Стрелка указывает на самую старую страницу.



Когда происходит страничное прерывание, поверяется страница, на которую указывает стрелка. Предпринимаемые действия зависят от бита R:

R=0: страница выгружается

R=1: бит R сбрасывается, стрелка движется вперед

При возникновении ошибки отсутствия страницы проверяется та страница, на которую указывает стрелка. Если ее бит *R* имеет значение 0, страница выселяется, на ее место в «циферблате» вставляется новая страница и стрелка передвигается вперед на одну позицию. Если значение бита *R* равно 1, то он сбрасывается и стрелка перемещается на следующую страницу. Этот процесс повторяется до тех пор, пока не будет найдена страница с $R = 0$.

Алгоритм замещения наименее востребованной страницы

В основе алгоритма лежит наблюдение, что страницы, интенсивно используемые несколькими последними командами, будут, скорее всего, снова востребованы следующими несколькими командами. И наоборот, долгое время не востребованные страницы наверняка еще долго так и останутся невостребованными. Эта мысль наталкивает на вполне реализуемый алгоритм: при возникновении ошибки отсутствия страницы нужно избавиться от той страницы, которая длительное время не была востребована. Эта стратегия называется замещением наименее востребованной страницы (Least Recently Used - **LRU**).

Алгоритма LRU сложно реализовать на аппаратном уровне. Одно из возможных решений на программном уровне называется алгоритмом **нечастого востребования** (Not Frequently Used - **NFU**). Для его реализации потребуется программный счетчик с начальным нулевым значением, связанный с каждой страницей. При каждом прерывании от таймера операционная система сканирует все находящиеся в памяти страницы. Для каждой страницы к счетчику добавляется значение бита *R*, равное 0 или 1. Счетчики позволяют приблизительно отследить частоту обращений к каждой странице. При возникновении ошибки отсутствия страницы для замещения выбирается та страница, чей счетчик имеет наименьшее значение.

Алгоритм «рабочий набор»

Набор страниц, который процесс использует в данный момент, известен как **рабочий набор**. Если в памяти находится весь рабочий набор, процесс будет работать, не вызывая многочисленных ошибок отсутствия страниц, пока не перейдет к другой фазе выполнения.

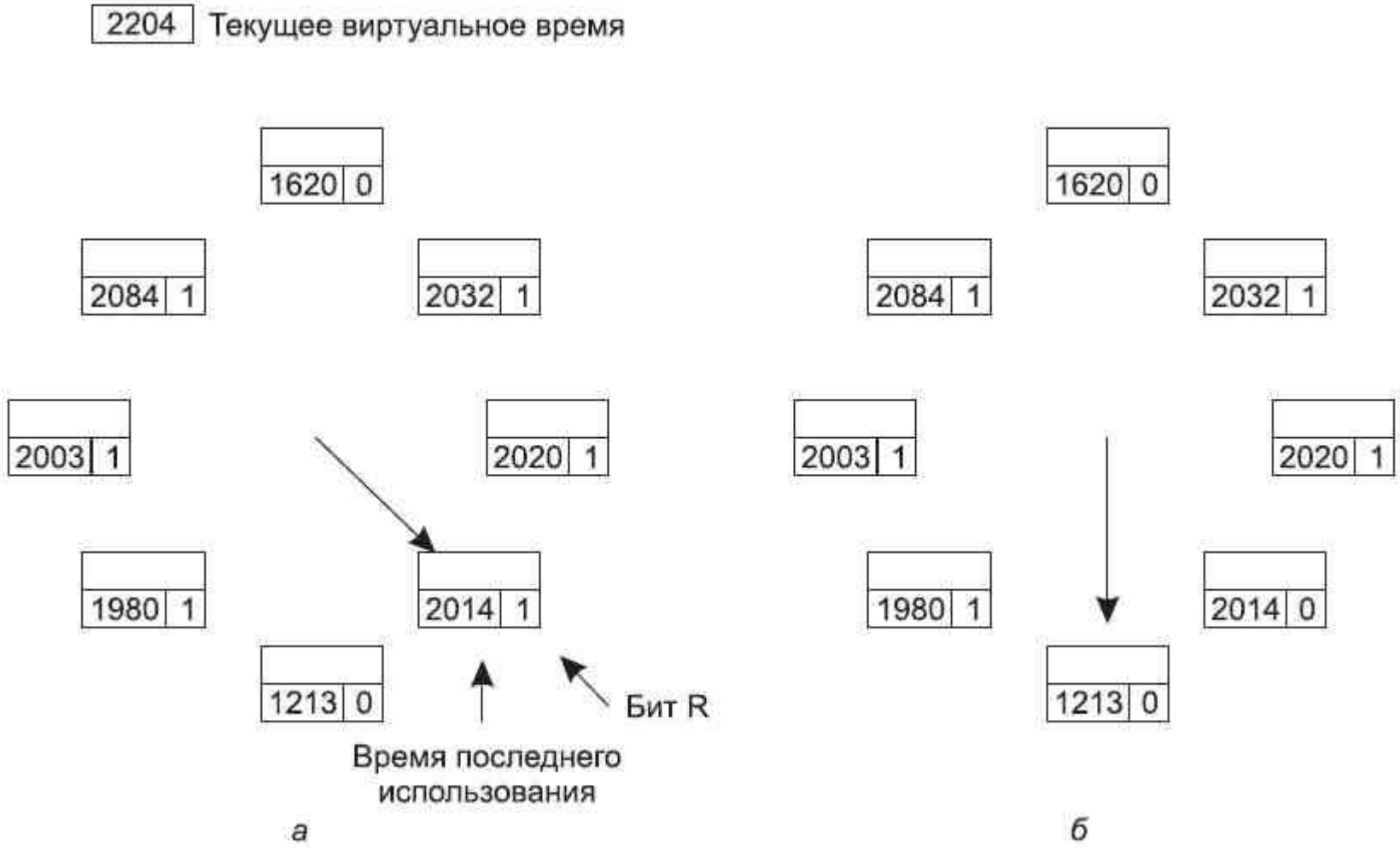
В многозадачных системах процессы довольно часто сбрасываются на диск (то есть все их страницы удаляются из памяти), чтобы дать возможность другим процессам воспользоваться своей очередью доступа к центральному процессору.

Поэтому многие системы замещения страниц пытаются отслеживать рабочий набор каждого процесса и обеспечивать его присутствие в памяти, перед тем как позволить процессу возобновить работу. Такой подход называется моделью рабочего набора. Он был разработан для существенного сокращения количества ошибок отсутствия страниц. Загрузка страниц *до того*, как процессу будет позволено возобновить работу, называется также опережающей подкачкой страниц (**prepaging**). Следует заметить, что со временем рабочий набор изменяется.

Алгоритм WSClock

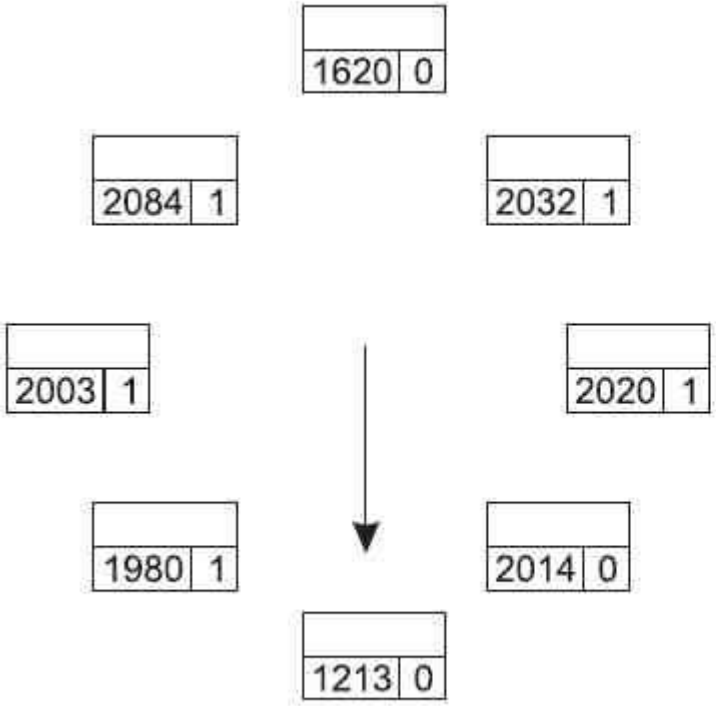
Базовый алгоритм рабочего набора слишком трудоемок, поскольку при возникновении ошибки отсутствия страницы для определения местонахождения подходящего кандидата на удаление необходимо просканировать всю таблицу страниц. Усовершенствованный алгоритм, основанный на алгоритме «часы», но также использующий информацию о рабочем наборе, называется **WSClock**. Благодаря простоте реализации и хорошей производительности он довольно широко используется на практике.

Необходимая структура данных сводится к циклическому списку страничных блоков, как в алгоритме «часы» и как показано на рис. *а*. Изначально этот список пуст. При загрузке первой страницы она добавляется к списку. По мере загрузки следующих страниц они попадают в список, формируя замкнутое кольцо. В каждой записи содержится поле *времени последнего использования* из базового алгоритма рабочего набора, а также бит *R* (показанный на рисунке) и бит *M* (не показанный на рисунке).


2204
Текущее виртуальное время
1620 0
2084 1
2032 1
2003 1
2020 1
1980 1
2014 1
1213 0
Бит R
Время последнего использования
а
1620 0
2084 1
2032 1
2003 1
2020 1
1980 1
2014 0
1213 0
б

Как и в алгоритме «часы», при каждой ошибке отсутствия страницы сначала проверяется страница, на которую указывает стрелка. Если бит *R* установлен в 1, значит, страница была использована в течение текущего такта, поэтому она не является идеальным кандидатом на удаление. Затем бит *R* устанавливается в 0, стрелка перемещается на следующую страницу, и алгоритм повторяется уже для нее. Состояние, получившееся после этой последовательности событий, показано на рис. *б*.

Теперь посмотрим, что получится, если у страницы, на которую указывает стрелка, бит $R = 0$ (рис. *в*). Если ее возраст превышает значение *t* и страница не изменена, она не относится к рабочему набору и ее точная копия присутствует на диске. Тогда в страничный блок просто помещается новая страница (рис. *г*). Но если страница изменена, ее блок не может быть тотчас же исползован, поскольку на диске нет ее точной копии. Чтобы избежать переключения процесса, запись на диск планируется, а стрелка перемещается дальше и алгоритм продолжает свою работу на следующей странице. В конце концов должна попасться старая, неизмененная страница, которой можно будет тут же и воспользоваться.


1620 0
2084 1
2032 1
2003 1
2020 1
1980 1
2014 0
1213 0

в


1620 0
2084 1
2032 1
2003 1
2020 1
1980 1
2014 0
2204 1
Новая страница

г

Резервное хранилище

Простейший алгоритм для выделения страничного пространства на диске предусматривает наличие на нем специального раздела подкачки (свопинга) или, что еще лучше, отделения дискового устройства от файловой системы. Подобным образом работает большинство UNIX-систем.

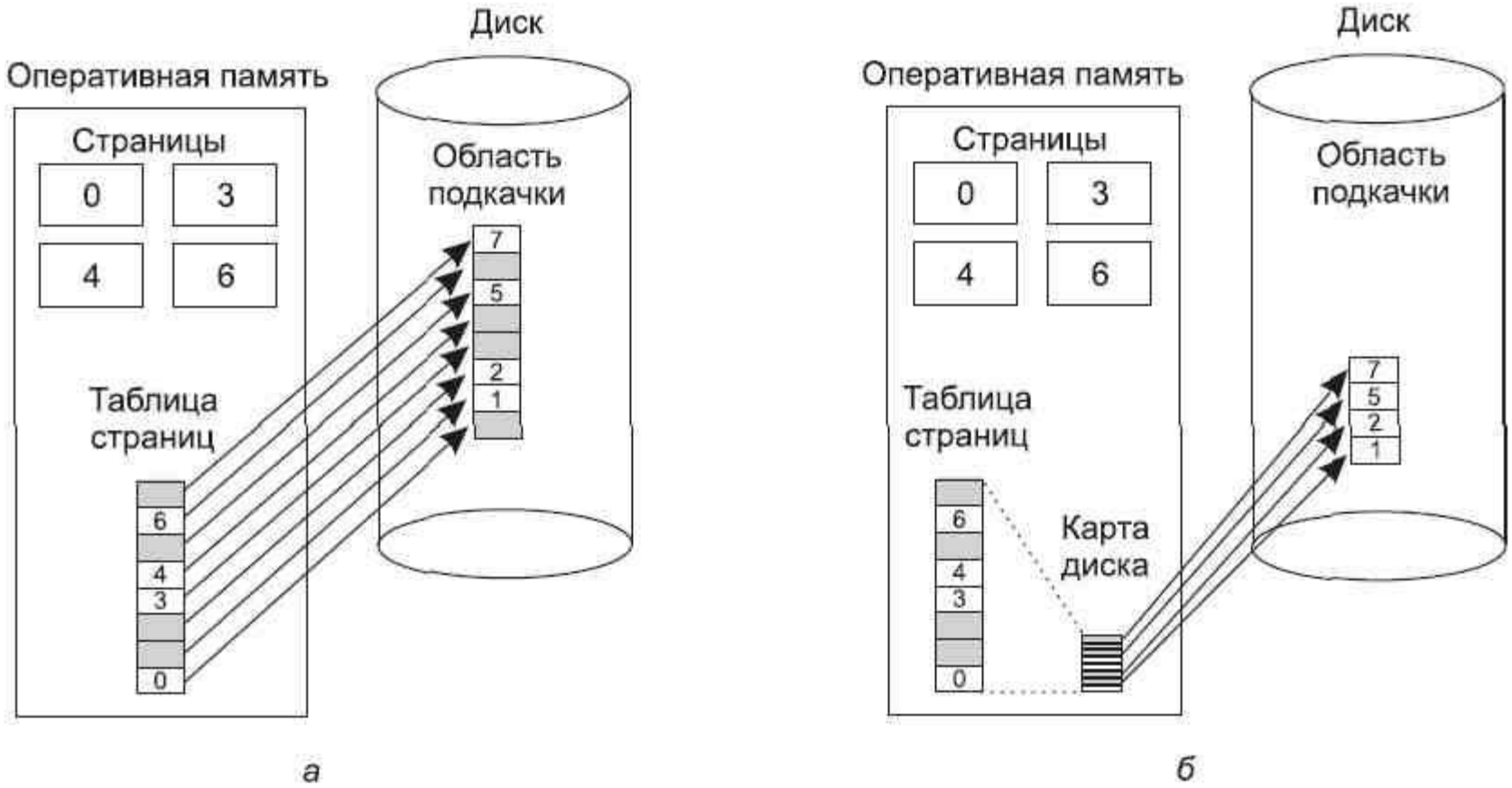
При запуске системы раздел подкачки пуст и представлен в памяти единой записью с указанием начального адреса и размера. По простейшей схеме при запуске первого процесса за ним резервируется участок пространства раздела, соответствующий размеру первого процесса, а оставшаяся область сокращается на эту величину. При запуске новых процессов им выделяется участок раздела подкачки, равный по размеру их образам. При завершении процессов их дисковые пространства освобождаются. Раздел подкачки управляется как список свободных участков.

С каждым процессом связывается дисковый адрес его области подкачки, то есть тот адрес, по которому в разделе подкачки хранится его образ. Эта информация хранится в таблице процессов. Но перед тем как процесс сможет начать работу, область свопинга должна быть инициализирована.

Один из способов инициализации заключается в копировании всего образа процесса в область свопинга, чтобы его можно было *получать* по мере необходимости. Другой способ заключается в загрузке всего процесса в память и разрешении ему по мере необходимости *выгружать* страницы.

Но с этой простой моделью связана одна проблема: размеры процессов после их запуска могут изменяться. Хотя размер текста программы обычно не меняется, область данных иногда может расти, а стек всегда склонен к росту. Следовательно, может быть лучше резервировать отдельные области подкачки для текста, данных и стека и давать возможность каждой из этих областей состоять более чем из одного дискового участка.

Другая крайность заключается в полном отказе от какого-либо предварительного распределения, выделении дискового пространства для каждой страницы при ее выгрузке на диск и его изъятии при обратной загрузке страницы в память. При этом находящиеся в памяти процессы вообще не привязываются к пространству свопинга. Недостаток такого способа заключается в необходимости хранения в памяти дискового адреса, чтобы отслеживать на диске каждую страницу. Эти два альтернативных варианта показаны на рисунке.


Диск
Оперативная память
Страницы
0
3
4
6
Область подкачки
7
5
2
1
Таблица страниц
6
4
3
0
Карта диска
а
б

На рис.*а* показана таблица страниц с восемью страницами. Страницы 0, 3, 4 и 6 находятся в оперативной памяти, страницы 1, 2, 5 и 7 — на диске. Размер области свопинга совпадает по размеру с виртуальным адресным пространством процесса (восемь страниц), а у каждой страницы есть фиксированное место, в которое она записывается при удалении из основной памяти. У страницы, находящейся в памяти, всегда есть ее копия на диске (закрашенная область), но эта копия может устареть, если страница с момента загрузки подверглась изменению. На рис. *а* в памяти закрашенными областями показаны отсутствующие в ней страницы. Страницы, соответствующие закрашенным областям, на диске должны быть заменены копиями в памяти, хотя, если страница памяти должна быть сброшена на диск и не подвергалась модификации со времени своей загрузки, то будет использована ее дисковая (закрашенная) копия.

У страниц, изображенных на рис. *б* нет фиксированных адресов на диске. При выгрузке страницы выбирается пустая дисковая страница и соответствующим образом обновляется карта диска. Страница в памяти не имеет своей копии на диске. Записи страниц на карте диска содержат либо неправильный адрес диска, либо бит, помечающий их как неиспользующиеся.

Возможность иметь фиксированный раздел подкачки предоставляется не всегда. Например, могут отсутствовать доступные дисковые разделы. В таком случае может использоваться один или несколько заранее выделенных файлов внутри обычной файловой системы. Именно такой подход используется в Windows.