## 1 Супервизор, его фунцкии и состав.

**Ядро супервизора** – совокупность программ (управляющие программы), обеспечивающих функционирование ВС и находящихся в системной области оперативной памяти, составляет ядро супервизора.

Состав:

1) Пр-мы управления заданиями. Отслеживает вход и выход задания в системе

2) Пр-мы управления задачами (процессами)

3) Система управления файлами.

4) Управление памятью. Делиться на организацию и управление памятью

5) Управление внешними устройствами

Следующие из этого ф-ции:

Обработка прерываний; создание и уничтожение процессов; переключение процессов из состояния в состояние; диспетчеризацию заданий, процессов и ресурсов; приостановка и активизация процессов; синхронизация процессов; организация взаимодействия между процессами; манипулирование РСВ; поддержка операций ввода/вывода; поддержка распределения и перераспределения памяти; поддержка работы файловой системы; поддержка механизма вызова – возврата при обращении к процедурам; поддержка определенных функций по ведению учета работы машины;

Одна из самых важных функций, реализованная в ядре – обработка прерываний.

**Супервизор**

В супервизоре принимаются решения по использованию ресурсов, необходимых для выполнения его многочисленных функций. В него входят менеджер реальной памяти, менеджер виртуальной памяти, менеджер ресурсов и менеджер вспомогательной памяти, В нем также содержатся программы, управляющие мультипрограм­мной работой.

Супервизор — обеспечивает управление мультипрограммирова­нием. Он создает диспетчеризуемую единицу работы, осущест­вляет переключения (диспетчеризация), обеспечивает последова­тельное использование ресурсов (например, предоставляет средст­ва, обеспечивающие взаимоисключение).

Место, где находится резиденция системы, называется " резидентным" гомом, а сама система "резиденцией". Из всех программных модулей в резидентном томе часть используется по мере необходимости, а некото­рые нз них обеспечивают базовое функционирование ВС и составляют *резидентные* программы, находящиеся в системной области оперативной памяти. Совокупность программ, обеспечивающих функционирование ВС и находящихся в системной области оперативной памяти, составляет ядро супервизора.

При загрузке ОС необходимо выполнить связывание, размещение всех частей, входящих у ядро супервизора, т.е. размещение резидентных программ на своих местах, формирование специальных системных струк­тур данных в области данных операционной системы, формирование по­стоянной области, активизацию процессов для начала работы операци­онной системы. Этот процесс называется *инициализацией* операционной системы.

Доп. Инфа

Как обрабатывается прерывание по обращению к супервизору.

*Прерывания по обращению к супервизору*. Вызываются при выполнении процессором **команды обращения к супервизору** (вызов функции операционной системы). Обычно такая команда инициируется выполняемым процессом при необходимости получения дополнительных ресурсов либо при взаимодействии с устройствами ввода/вывода.

**Структура ядра супервизора.**

Совокупность программ, обеспечивающих функционирование ВС и находящихся в системной области оперативной памяти, составляет ядро супервизора.

Структура ядра:

-- резидентные программы

-- системные таблицы

## 2 Потоки и особенности их применения при организации вычислений

Внутри процесса могут сосущестовать несколько параллельных потоков (нитей). Каждый из таких потоков является по сути "процессом в процессе", но имеет существенное отличие - поток не имеет собственных ресурсов. Поэтому потоки делят между собой адресное пространства процесса, в котором они были порождены. Потоки могут успешно применятся как для обеспечения кажущегося совмещения, так и для истинного параллелизма в мультипроцессорных системах. Поскольку потоки делят адресное пространство своего процесса, необходимо следить за выполнением синхронизации и взаимного исключения, если потоки зависят друг от друга и используют общие данные.

## Теория рабочего множества и ее влияние на эффективность систем управления памятью. Особенности реализации в современных ВС.

Понятие окна рабочего множества.

Процессы начинают работать, не имея в памяти необходимых страниц. В результате при выполнении первой же машинной инструкции возникает page fault, требующий подкачки порции кода. Следующий page fault происходит при локализации глобальных переменных и еще один - при выделении памяти для стека. После того как процесс собрал большую часть необходимых ему страниц, page faults возникают редко.

Таким образом, существует набор страниц (P1, P2, ... Pn), активно использующихся вместе, который позволяет процессу в момент времени t в течение некоторого периода T производительно работать, избегая большого количества page faults. Этот набор страниц называется рабочим множеством W(t,T) (working set) процесса. Число страниц в рабочем множестве определяется параметром Т, является неубывающей функцией T и относительно невелико. Иногда T называют размером окна рабочего множества, через которое ведется наблюдение за процессом (см. рис. 10.3).

Рис. 10.3. Пример рабочего множества процесса

Легко написать тестовую программу, которая систематически работает с большим диапазоном адресов, но, к счастью, большинство реальных процессов не ведут себя подобным образом, а проявляют свойство локальности. В течение любой фазы вычислений процесс работает с небольшим количеством страниц.

Когда процесс выполняется, он двигается от одного рабочего множества к другому. Программа обычно состоит из нескольких рабочих множеств, которые могут перекрываться. Hапример, когда вызвана процедура, она определяет новое рабочее множество, состоящее из страниц, содержащих инструкции процедуры, ее локальные и глобальные переменные. После ее завершения процесс покидает это рабочее множество, но может вернуться к нему при новом вызове процедуры. Таким образом, рабочее множество определяется кодом и данными программы. Если процессу выделять меньше кадров, чем ему требуется для поддержки рабочего множества, он будет находиться в состоянии трешинга.

Принцип локальности ссылок препятствует частым изменениям рабочих наборов процессов. Формально это можно выразить следующим образом. Если в период времени (t-T, t) программа обращалась к страницам W(t,T), то при надлежащем выборе T с большой вероятностью эта программа будет обращаться к тем же страницам в период времени (t, t+T). Другими словами, принцип локальности утверждает, что если не слишком далеко заглядывать в будущее, то можно достаточно точно его прогнозировать исходя из прошлого. Понятно, что с течением времени рабочий набор процесса может изменяться (как по составу страниц, так и по их числу).

Наиболее важное свойство рабочего множества - его размер. ОС должна выделить каждому процессу достаточное число кадров, чтобы поместилось его рабочее множество. Если кадры еще остались, то может быть инициирован другой процесс. Если рабочие множества процессов не помещаются в память и начинается трешинг, то один из процессов можно выгрузить на диск.

Алгоритм «Рабочий набор»

При использовании замещения страниц в простейшей форме процессы начинают свою работу, не имея в памяти вообще никаких страниц. Как только центральный процессор пытается извлечь первую команду, он получает ошибку отсутствия страницы, заставляющую операционную систему ввести в память страницу, содержащую первую команду. Обычно вскоре за этим следуют ошибки отсутствия страниц с глобальными переменными и стеком. Через некоторое время процесс располагает большинством необходимых ему страниц и приступает к работе, сталкиваясь с ошибками отсутствия страниц относительно редко. Эта стратегия называется замещением страниц по требованию (demand paging) поскольку страницы загружаются только по мере надобности, а не заранее.

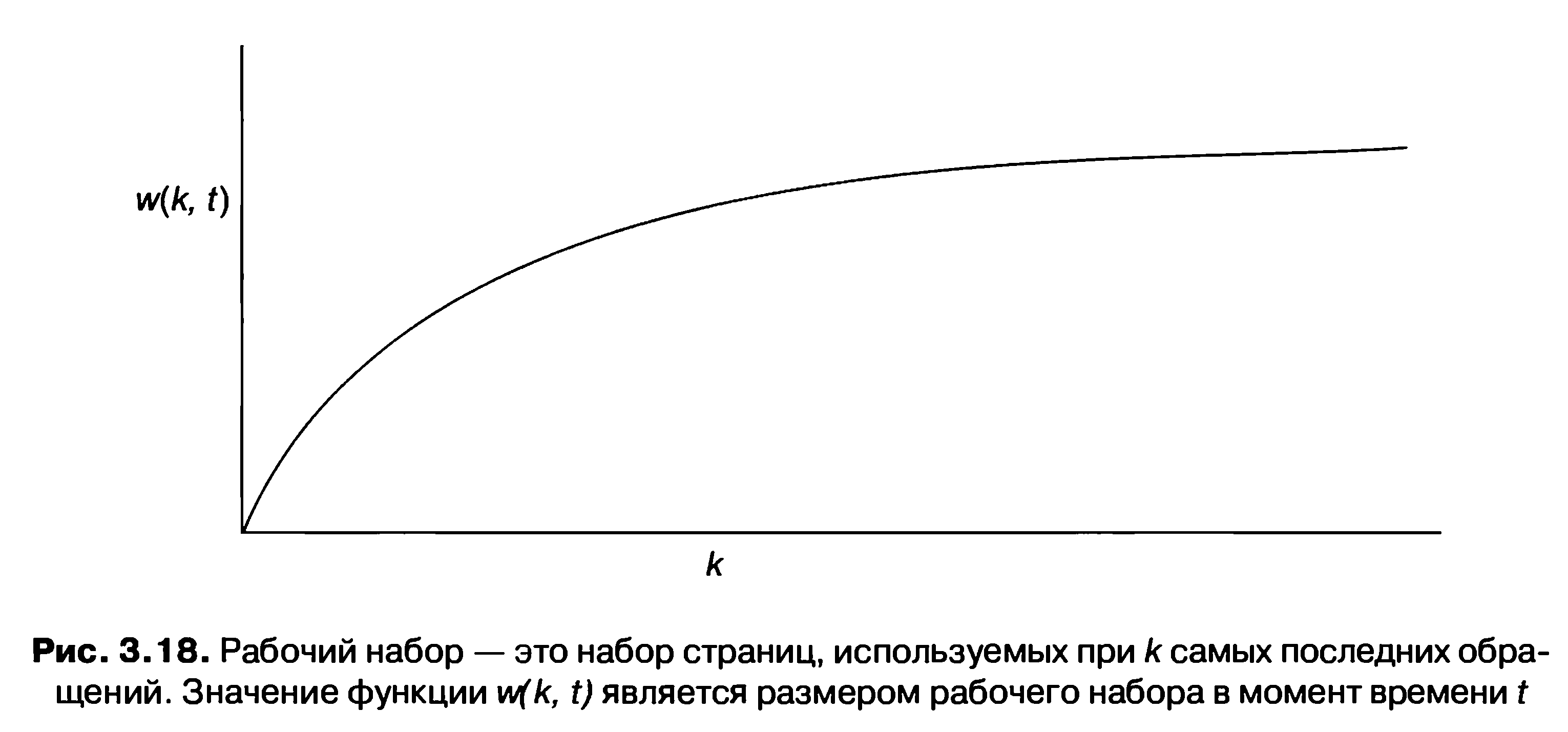
Разумеется, нетрудно написать тестовую программу, систематически читающую все страницы в огромном адресном пространстве, вызывая при этом такое количество ошибок отсутствия страниц, что для их обработки не хватит памяти. К счастью, большинство процессов так не работают. Ими применяется локальность обращений, означающая, что в течение любой фазы выполнения процесс обращается только к относительно небольшой части своих страниц. К примеру, при каждом проходе многопроходного компилятора обращение идет только к части имеющихся страниц, причем всякий раз к другой части.

Набор страниц, использующихся процессом в данный момент, известен как рабочий набор (Denning, 1968a; Denning, 1980). Если в памяти находится весь рабочий набор, процесс будет работать, не вызывая многочисленных ошибок отсутствия страниц, пока не перейдет к другой фазе выполнения (например, к следующему проходу компилятора). Если объем доступной памяти слишком мал для размещения всего рабочего набора, процесс вызовет множество ошибок отсутствия страниц и будет работать медленно, поскольку выполнение команды занимает всего несколько наносекунд, а считывание страницы с диска обычно занимает 10 мс. Если он будет выполнять одну или две команды за 10 мс, то завершение его работы займет целую вечность. Про программу, вызывающую ошибку отсутствия страницы через каждые несколько команд, говорят, что она пробуксовывает (Denning, 19686).

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. Возникает вопрос, что делать, когда процесс возобновляет свою работу С технической точки зрения ничего делать не нужно. Процесс просто будет вызывать ошибки отсутствия страниц до тех пор, пока не будет загружен его рабочий набор. Проблема в том, что наличие 20, 100 или даже 1000 ошибок отсутствия страниц при каждой загрузке процесса замедляет работу, а также вызывает пустую трату значительной части рабочего времени центрального процессора, поскольку на обработку операционной системой одной ошибки отсутствия страницы затрачивается несколько миллисекунд процессорного времени.

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 (Denning, 1970). 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.

Давно подмечено, что большинство программ неравномерно обращается к своему адресному пространству, их обращения склонны группироваться на небольшом количестве страниц. Обращение к памяти может быть извлечением команды, извлечением данных или сохранением данных. В любой момент времени t существует некий набор, состоящий из всех страниц, используемый в k самых последних обращений к памяти. Этот набор, w(k, t), и является рабочим набором. Так как все недавние обращения к памяти для k > 1 обязательно должны были обращаться ко всем страницам, использовавшимся для k = 1 обращения к памяти, то есть к последней и, возможно, еще к некоторым страницам, w(k, t) является монотонно неубывающей функцией от к По мере роста значения k значение функции w(k, t) достигает своего предела, поскольку программа не может обращаться к количеству страниц, превышающему по объему имеющееся адресное пространство, а каждая отдельно взятая страница будет использоваться лишь немногими программами. На рис. 3.18 показано, что размер рабочего набора является функцией от к.



Тот факт, что большинство программ произвольно обращается к небольшому количеству страниц, но со временем этот набор медленно изменяется, объясняет начальный быстрый взлет кривой графика, а затем, при больших значениях к, замедление этого взлета. К примеру, программа, выполняющая цикл, при этом занимающая две страницы и использующая данные, расположенные на четырех страницах, может обращаться ко всем шести страницам каждые 1000 команд, но самые последние обращения к некоторым другим страницам могут состояться за миллион команд до этого, в процессе фазы инициализации. Благодаря такому асимптотическому поведению содержимое рабочего набора нечувствительно к выбранному значению к.

Иначе говоря, существует широкий диапазон значений к, для которого рабочий набор остается неизменным. Поскольку со временем рабочий набор изменяется медленно, появляется возможность выстроить разумные предположения о том, какие страницы понадобятся при возобновлении работы программы, основываясь на том, каков был ее рабочий набор при последней приостановке ее работы. Опережающая подкачка страниц как раз и заключается в загрузке этих страниц перед возобновлением процесса.

Для реализации модели рабочего набора необходимо, чтобы операционная система отслеживала, какие именно страницы входят в рабочий набор. При наличии такой информации тут же напрашивается и возможный алгоритм замещения страниц: при возникновении ошибки отсутствия страницы нужно выселить ту страницу, которая не относится к рабочему набору. Для реализации подобного алгоритма нам необходим четкий способ определения, какие именно страницы относятся к рабочему набору. По определению рабочий набор — это набор страниц, используемых в к самых последних обращений (некоторые авторы используют термин к самых последних страничных обращений, но это дело вкуса). Для реализации любого алгоритма рабочего набора некоторые значения к должны быть выбраны заранее. Как только после каждого обращения к памяти будет выбрано некоторое значение, однозначно определяется и набор страниц, используемый при самых последних к обращениях к памяти.

Разумеется, имеющееся определение рабочего набора не означает наличия эффективного способа его вычисления в процессе выполнения программы. Можно представить себе регистр со сдвигом, имеющий длину k, в котором при каждом обращении к памяти его содержимое сдвигается влево на одну позицию и справа вставляется номер страницы, к которой было самое последнее обращение. Набор из всех k номеров страниц в регистре со сдвигом и будет представлять собой рабочий набор. Теоретически при возникновении ошибки отсутствия страницы содержимое регистра со сдвигом может быть считано и отсортировано. Затем могут быть удалены продублированные страницы. В результате должен получиться рабочий набор. Но обслуживание регистра со сдвигом и обработка его содержимого при возникновении ошибки отсутствия страницы окажется недопустимо затратным делом, поэтому эта технология никогда не используется. Вместо нее используются различные приближения. Одно из часто используемых приближений сводится к отказу от идеи вычисления k обращений к памяти и использованию вместо этого времени выполнения. Например, вместо определения рабочего набора в качестве страниц, использовавшихся в течение предыдущих 10 миллионов обращений к памяти, мы можем определить его как набор страниц, используемых в течение последних 100 мс времени выполнения.

На практике с таким определением работать гораздо лучше и проще. Следует заметить, что для каждого процесса вычисляется только его собственное время выполнения. Таким образом, если процесс запускается во время Т и получает 40 мс времени центрального процессора за время Т+ 100 мс, то для определения рабочего набора берется время 40 мс. Интервал времени центрального процессора, реально занимаемый процессом с момента его запуска, часто называют текущим виртуальным временем. При этом приближении рабочим набором процесса

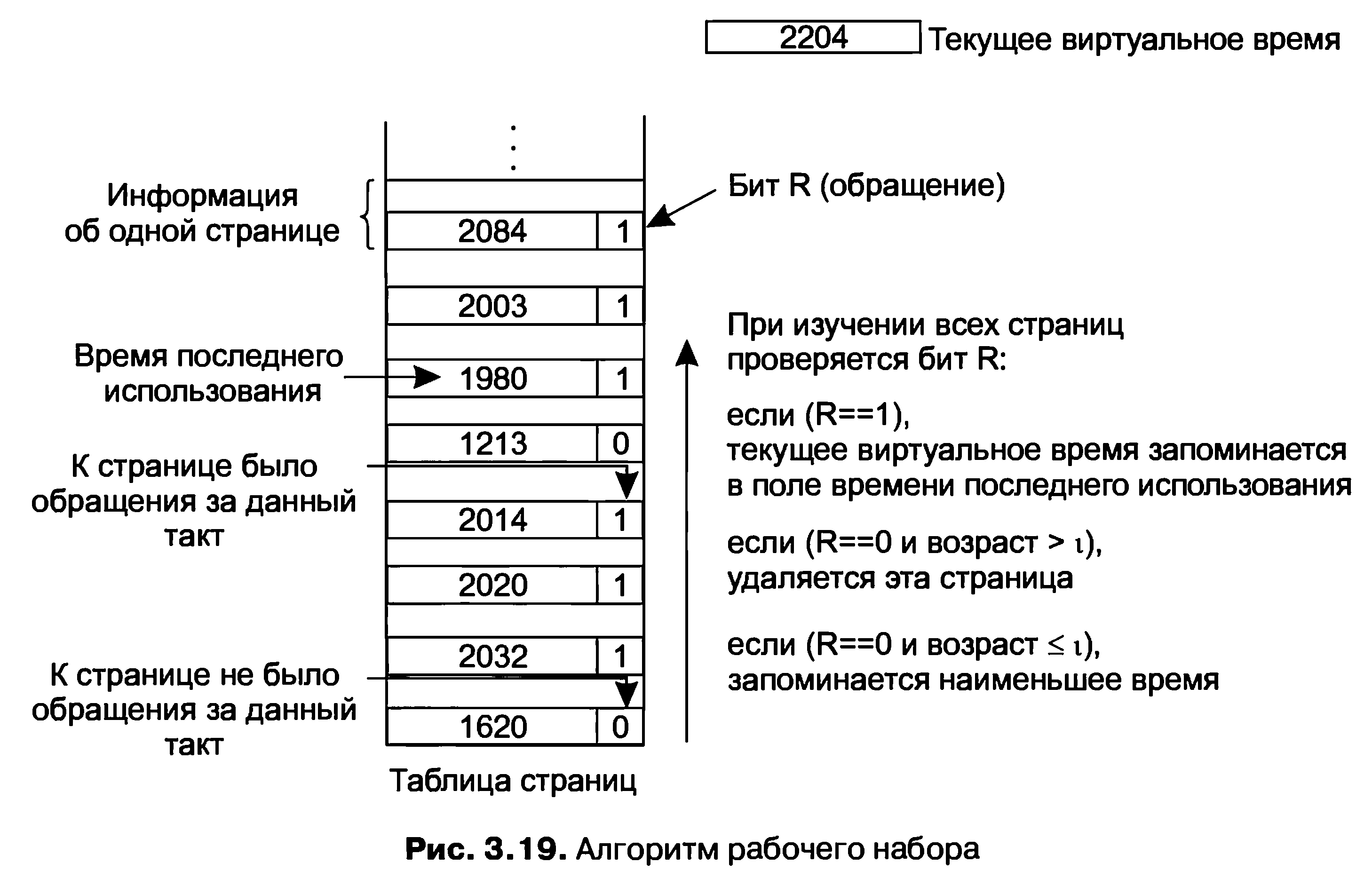
является набор страниц, к которым он обращался в течение последних т секунд виртуального времени. Теперь взглянем на алгоритм замещения страниц, основанный на рабочем наборе. Основной замысел состоит в том, чтобы найти страницу, не принадлежащую рабочему набору, и удалить ее из памяти. На рис. 3.19 показана часть таблицы страниц, используемой в некой машине. Поскольку кандидатами на выселение рассматриваются только страницы, находящиеся в памяти, страницы, в ней отсутствующие, этим алгоритмом игнорируются. Каждая запись состоит (как минимум) из двух ключевых элементов информации: времени (приблизительного) последнего использования страницы и бита R (Referenced — бита обращения).

Пустыми белыми прямоугольниками обозначены другие поля, не нужные для этого алгоритма, например номер страничного блока, биты защиты и бит изменения — М (Modified).

Рассмотрим работу алгоритма. Предполагается, что аппаратура, как это было рассмотрено ранее, устанавливает биты Rn M. Также предполагается, что периодические прерывания от таймера запускают программу, очищающую бит обращения R. При каждой ошибке отсутствия страницы происходит сканирование таблицы страниц с целью найти страницу, пригодную для удаления.

При каждой обработке записи проверяется состояние бита R. Если его значение равно 1, текущее виртуальное время записывается в поле времени последнего использования таблицы страниц, показывая, что страница была использована при возникновении ошибки отсутствия страницы. Если обращение к странице происходит в течение текущего такта времени, становится понятным, что она принад лежит рабочему набору и не является кандидатом на удаление (предполагается,

что т охватывает несколько системных тактов).



Если значение R равно 0, значит, за текущий такт времени обращений к странице не было и она может быть кандидатом на удаление. Чтобы понять, должна ли она быть удалена или нет, вычисляется ее возраст (текущее виртуальное время за вычетом ее времени последнего использования), который сравнивается со значением т. Если возраст превышает значение т, то страница уже не относится к рабочему набору и заменяется новой страницей. Сканирование продолжается, и происходит обновление всех остальных записей.

Но если значение R равно 0, но возраст меньше или равен т, то страница все еще относится к рабочему набору. Страница временно избегает удаления, но страница с наибольшим возрастом (наименьшим значением времени последнего использования) берется на заметку Если будет просканирована вся таблица страниц и не будет найдена страница-кандидат на удаление, значит, к рабочему набору относятся все страницы. В таком случае, если найдена одна и более страниц с R = 0, удаляется одна из них, имеющая наибольший возраст. В худшем случае в течение текущего

такта было обращение ко всем страницам (и поэтому у всех страниц R = 1), поэтому для удаления одна из них выбирается случайным образом, при этом предпочтение отдается неизмененной странице, если таковая имеется.

## Принципы повышения эффективности файловых систем

Кэширование

Для минимизации к-ва обращений к диску применяется блочный кэш или буферный кэш (набор блоков, логически принадлежащих диску, но хранящихся в ОП).

Перехватка всех запросов чтения к диску и проверке наличия требующихся блоков в кэше. Если блок присутствует в кэше, то запрос чтения блока может быть удовлетворен без обращения к диску. В противном случае блок сначала считывается с диска в кэш, а оттуда копируется по нужному адресу памяти. По следующие обращения к тому же блоку могут удовлетворяться из кэша.

Опережающее чтение блока

Получение блоков диска в кэш прежде, чем они потребуются. Многие файлы считываются последовательно. Когда файловая система получает запрос на чтение блока к файла, она выполняет его, но после этого сразу проверя ет, есть ли в кэше блок к + 1. Если этого блока в кэше нет, файловая система читает его в надежде, что к тому моменту, когда он понадобится, этот блок уже будет счи тан в кэш. В крайнем случае, он уже будет на пути туда.

Если обращения к блокам файла производятся в случайном по рядке, опережающее чтение не помогает.

Снижение времени перемещения блока головок

Другой важный метод состоит в уменьше нии затрат времени на перемещение блока головок. Достигается это помещением блоков, к которым высока вероятность доступа в течение короткого интервала вре мени, близко друг к другу, желательно на одном цилиндре. Когда записывается выходной файл, файловая система должна зарезервировать место для чтения та ких блоков за одну операцию. Если свободные блоки учитываются в битовом мас сиве, а весь битовый массив помещается в оперативной памяти, то довольно легко выбрать свободный блок как можно ближе к предыдущему блоку. В случае когда свободные блоки хранятся в списке, часть которого в оперативной памяти, а часть на диске, сделать это значительно труднее.

## Фазы прерываний и их особенности

В процессе обработки прерывания можно выделить следующие фазы прерывания (рис. 2.20):

Рис. 2.20. Фазы прерывания

1. Тз — время задержки между моментом возникновения сигнала прерывания и прерыванием активного процесса. Оно зависит от принятого в системе (процессоре) способа обработки сигнала прерывания (смотри выше).

2. Тс — время сохранения необходимой информации. Зависит от количества сохраняемой информации при принятом способе обработки сигнала прерывания.

3. Тд — время дешифрации сигнала прерывания. Зависит от аппаратуры, дешифрирующей сигнал прерывания.

4. Тв — время восстановления прерванного процесса. Зависит от количества восстанавливаемой информации.

Время между возникновением сигнала прерывания и началом выполнения обработчика прерывания называется временем реакции системы на сигнал прерывания (Тр).

Для реализации механизма прерываний необходима аппаратная схема фиксации сигнала запроса на прерывание. Такая схема обычно содержит регистр, на котором фиксируется наличие сигналов во входных линиях запросов на прерывания. Объединенные схемой "ИЛИ", сигналы с разрядов регистра формируют общий сигнал о наличии запроса на прерывание. Затем все разряды регистра опрашиваются в порядке приоритетов входных линий. При этом используются 2 варианта реализации опроса:

1. Полноупорядоченная схема. Регистры опрашиваются по порядку приоритетов, от высшего к низшему. Это простая схема, однако при возникновении прерываний с низким приоритетом необходимо проверить все регистры запросов на прерывания с более высоким приоритетом.

2. Частично упорядоченная схема. Все прерывания делятся на классы и вводится 2-уровневая система приоритетов: первый уровень — среди различных классов, второй — внутри каждого класса. При этом, сначала по полноупорядоченной схеме определяется класс прерывания, на которое поступил запрос, а затем, также по полноупорядоченной схеме внутри установленного класса, определяется само прерывание. Это ускоряет поиск прерывания с низким приоритетом, однако усложняет процедуру поиска.