Алгоритмы выборов

Алгоритмы голосования

- Многие распределенные алгоритмы требуют, чтобы один из процессов был координатором, инициатором или выполнял другую специальную роль.
- Обычно не важно, какой именно процесс выполняет эти специальные действия, главное, чтобы он вообще существовал.
- Роль координатора может выполнять любой процесс.
- Любой процесс может инициировать процедуру выборов.
- Каждый процесс имеет некоторый уникальный номер.
- В общем, алгоритмы голосования пытаются найти процесс с максимальным номером и назначить его координатором. Алгоритмы различаются способами поиска координатора.

Алгоритм забияки

- Когда один из процессов замечает, что координатор больше не отвечает на запросы, он инициирует голосование. Процесс, например P, проводит голосование следующим образом.

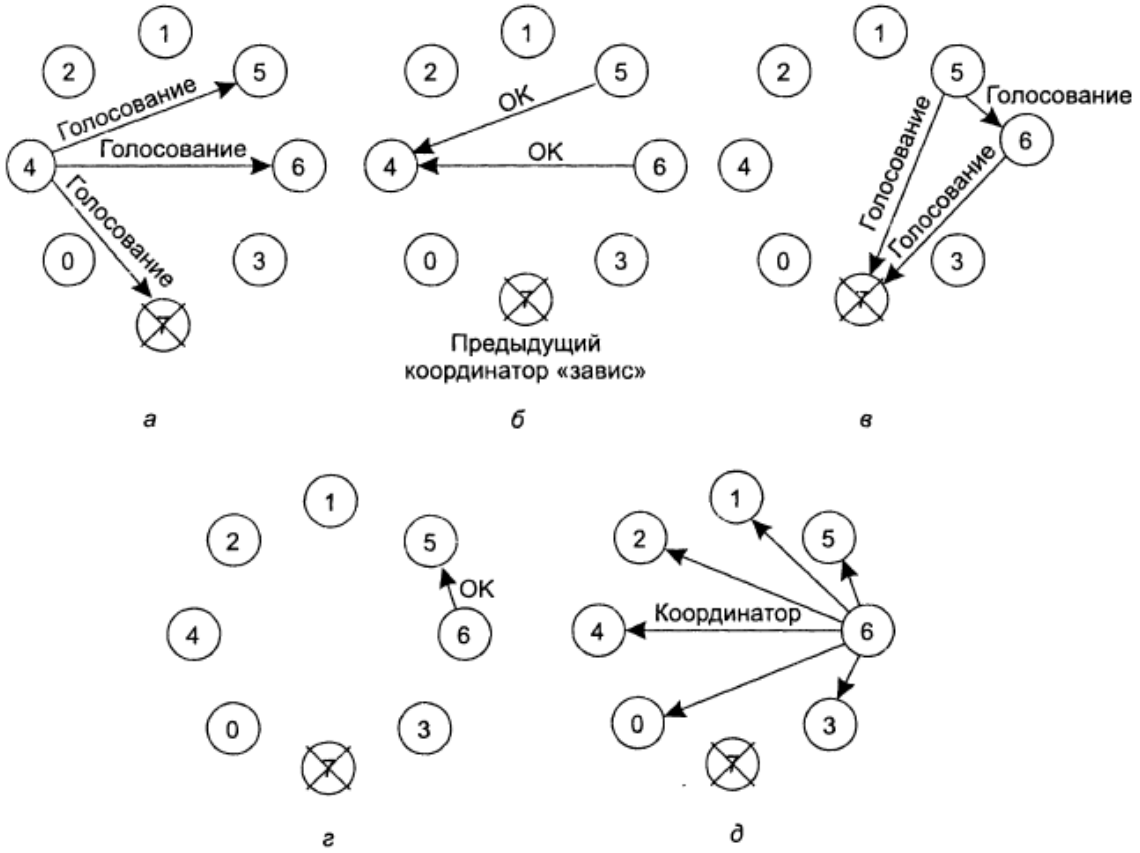
  1. P посылает всем процессам с большими, чем у него, номерами сообщение ГОЛОСОВАНИЕ.

  2. Далее возможно два варианта развития событий:

  - если никто не отвечает, P выигрывает голосование и становится координатором;
  - если один из процессов с большими номерами отвечает, он становится координатором, а работа P на этом заканчивается.

- В любой момент процесс может получить сообщение ГОЛОСОВАНИЕ от одного из своих коллег с меньшим номером. По получении этого сообщения получатель посылает отправителю сообщение ОК, показывая, что он работает и готов стать координатором.
- Затем получатель сам организует голосование. В конце концов, все процессы, кроме одного, отпадут, этот последний и будет новым координатором. Он уведомит о своей победе посылкой всем процессам сообщения, гласящего, что он новый координатор и приступает к работе.

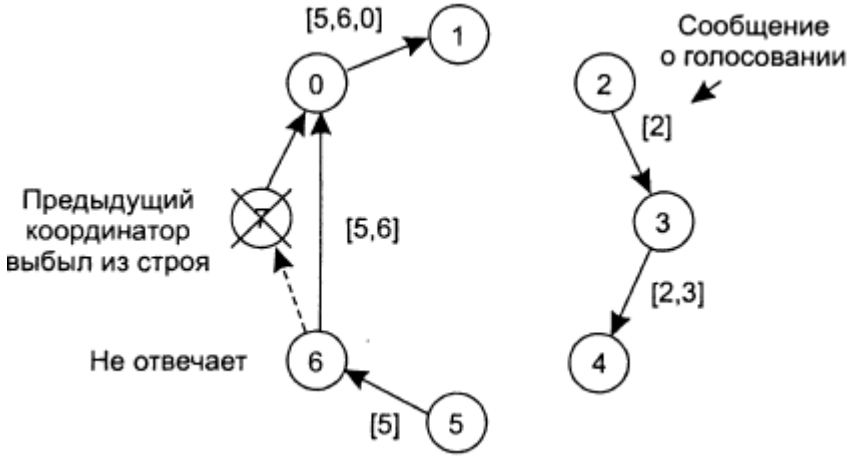
Голосование по алгоритму забияки



- Ранее координшатором был процесс 7, но он завис.
- Процесс 4 первым замечает это и посылает сообщение *ГОЛОСОВАНИЕ всем процессам с номерами больше, чем у него, то есть процессам*5, 6 и 7(*а).*
- Процессы 5 и 6 отвечают *ОК (б)*
- Оставшиеся процессы, 5 и 6, продолжают голосование (в).
- Каждый посылает сообщения только тем процессам, номера у которых
- процесс 6 сообщает процессу 5, что голосование будет вести он (г).
- В это время 6 понимает, что процесс 7 мертв, а значит, победитель — он сам (д).

Кольцевой алгоритм

- Этот алгоритм голосования основан на использовании логического кольца - процессы физически или логически упорядочены, так что каждый из процессов знает, кто его преемник.



- Когда два процесса, 2 и 5, одновременно обнаруживают, что предыдущий координатор, процесс 7, перестал работать.
- Каждый из них строит сообщение *ГОЛОСОВАНИЕ и запускает это сообщение* в путь по кольцу независимо от другого.
- Оба выбирают 6: [5,6,0,1,2,3,4] [2,3,4,5,6,0,1]

- Когда один из процессов обнаруживает, что координатор не функционирует, он строит сообщение *ГОЛОСОВАНИЕ, содержащее* его номер процесса, и посылает его своему преемнику.
- Если преемник не работает, отправитель пропускает его и переходит к следующему элементу кольца или к следующему, пока не найдет работающий процесс.
- На каждом шаге отправитель добавляет свой номер процесса к списку в сообщении, активно продвигая себя в качестве кандидата в координаторы.
- В конце концов, сообщение вернется к процессу, который начал голосование.
- В этот момент тип сообщения изменяется на *КООРДИНАТОР и вновь* отправляется по кругу, на этот раз с целью сообщить всем процессам, кто стал координатором (элемент списка с максимальным номером) и какие процессы входят в новое кольцо.

Взаимное исключение

Понятие критической области при работе с ресурсами

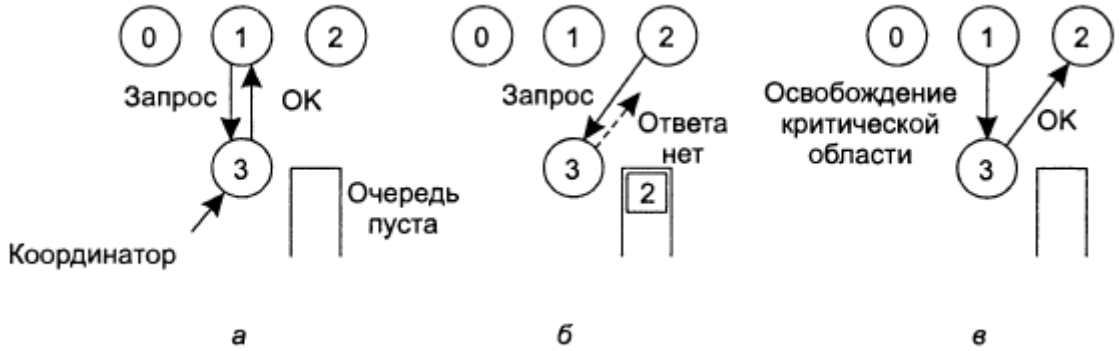
- Системы, состоящие из множества процессов, обычно проще всего программировать, используя критические области.
- Когда процесс нуждается в том, чтобы считать или обновить совместно используемые структуры данных, он сначала входит в критическую область, чтобы путем взаимного исключения убедиться, что ни один из процессов не использует одновременно с ним общие структуры данных.
- В однопроцессорных системах критические области защищаются семафорами, мониторами и другими конструкциями подобного рода.

Распределенные транзакции

- Концепция транзакций тесно связана с концепцией взаимных исключений.
- Алгоритмы взаимного исключения обеспечивают одновременный доступ не более чем одного процесса к совместно используемым ресурсам.
- Транзакции, в общем, также защищают общие ресурсы от одновременного доступа нескольких параллельных процессов.
- Однако транзакции могут и многое другое:
  - Они превращают процессы доступа и модификации множества элементов данных в одну атомарную операцию;
  - Если процесс во время транзакции решает остановиться на полпути и повернуть назад, все данные восстанавливаются с теми значениями и в том состоянии, в котором они были до начала транзакции.

Взаимные исключения в распределенных системах

- Наиболее простой способ организации взаимных исключений в распределенных системах состоит в том, чтобы использовать методы их реализации, принятые в однопроцессорных системах.
- Один из процессов выбирается координатором (например, процесс, запущенный на машине с самым большим сетевым адресом).
- Каждый раз, когда этот процесс собирается войти в критическую область, он посылает координатору сообщение с запросом, в котором уведомляет, в какую критическую область он собирается войти, и запрашивает разрешение на это. (а).



➢ Другой процесс, 2, запрашивает разрешение на вход в ту же самую область (*б). Координатор знает, что в этой критической* области уже находится другой процесс, и не дает разрешения на вход.

➢ Когда процесс 1 выходит из критической области, он сообщает об этом координатору, который разрешает доступ процессу 2 (в).

Распределенный алгоритм (1)

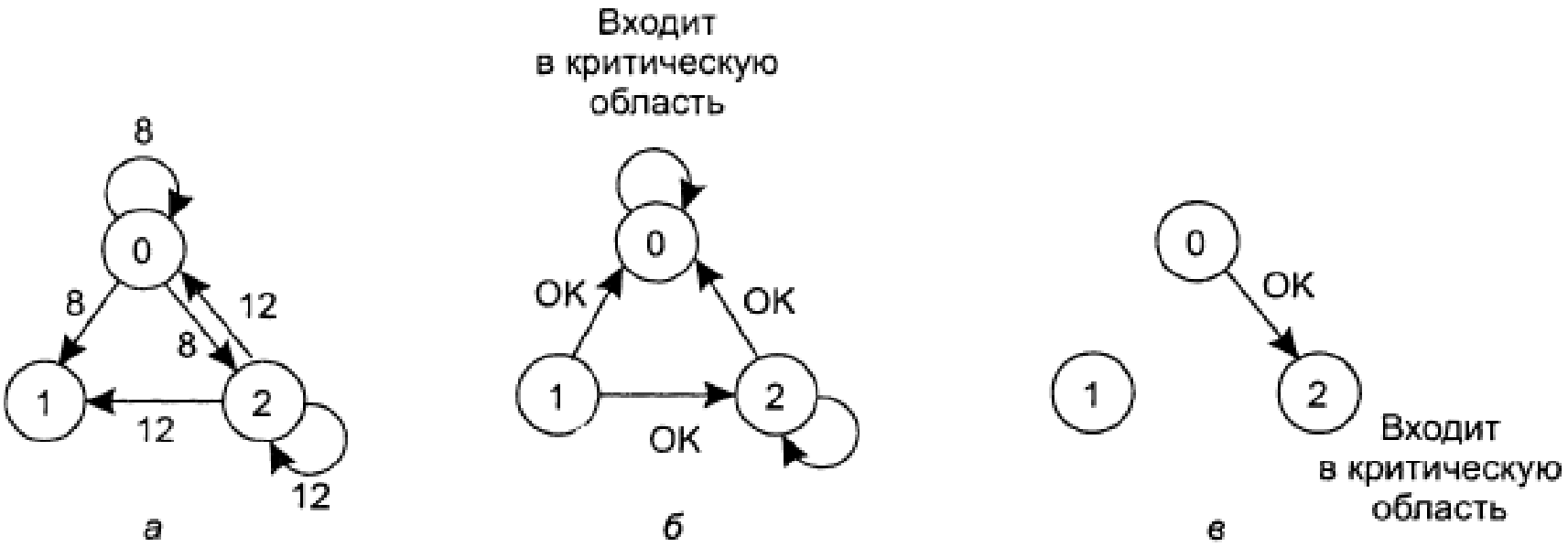
- Рассматриваемый алгоритм требуют наличия полной упорядоченности событий в системе. То есть в любой паре событий, например отправки сообщений, должно быть однозначно известно, какое из них произошло первым. Алгоритм Лампорта, является одним из способов введения подобной упорядоченности и может быть использован для расстановки отметок времени распределенных взаимных исключений.
- Когда процесс собирается войти в критическую область, он создает сообщение, содержащее имя критической области, свой номер и текущее время. Затем он отсылает это сообщение всем процес-сам, концептуально включая самого себя. Посылка сообщения, как предполагается, надежная, то есть на каждое письмо приходит подтверждение в получении. Вместо отдельных сообщений может быть использована доступная надежная групповая связь.
- Когда процесс получает сообщение с запросом от другого процесса, действие, которое оно производит, зависит от его связи с той критической областью, имя которой указано в сообщении.

Распределенный алгоритм (продолжение)

- Можно выделить три варианта:
  - Если получатель не находится в критической области и не собирается туда входить, он отсылает отправителю сообщение *ОК,*
  - *Если получатель находится в критической области, он не отвечает, а помещает* запрос в очередь.
  - Если получатель собирается войти в критическую область, но еще не сделал этого, он сравнивает метку времени пришедшего сообщения с меткой времени сообщения, которое он отослал. Выигрывает минимальное. Если пришедшее сообщение имеет меньший номер, получатель отвечает посылкой сообщения *ОК. Если его собственное сообщение имеет меньшую отметку* времени, получатель ставит приходящие сообщения в очередь, ничего не посылая при этом.
- После посылки сообщения-запроса на доступ в критическую область процесс приостанавливается и ожидает, что кто-нибудь даст ему разрешение на доступ. После того как все разрешения получены, он может войти в критическую область.
- Когда он покидает критическую область, то отсылает сообщения *ОК всем* процессам в их очереди и удаляет все сообщения подобного рода из своей очереди.

Работа алгоритма распределенного исключения

- Представим себе, что два процесса пытаются одновременно войти в одну и ту же критическую область (а).
- Процесс 0 имеет меньшую отметку времени и потому выигрывает (б).
- Когда процесс 0 завершает работу с критической областью, он отправляет сообщение ОК, и теперь процесс 2 может войти в критическую область *(в)*

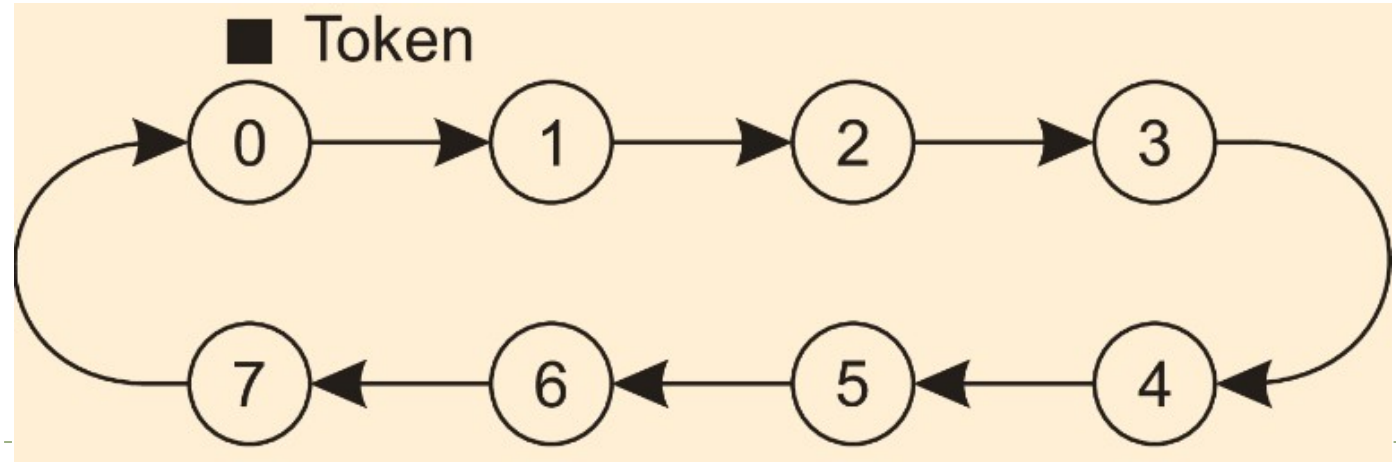


Проблемы распределенного алгоритма

- Если какой-либо из процессов «рухнет», он не сможет ответить на запрос. Это молчание будет воспринято (неправильно) как отказ в доступе и блокирует все последующие попытки всех процессов войти в какую-либо из критических областей.
- Этот алгоритм может быть исправлен так, что когда приходит запрос, его получатель посылает ответ всегда, разрешая или запрещая доступ. Всякий раз, когда запрос или ответ утеряны, отправитель выжидает положенное время и либо получает ответ, либо считает, что получатель находится в нерабочем состоянии. После получения запрещения отправитель ожидает последующего сообщения *OK*
- Другая проблема этого алгоритма состоит в том, что либо должны использоваться примитивы групповой связи, либо каждый процесс должен поддерживать список группы самостоятельно, обеспечивая внесение процессов в группу, удаление процессов из группы и отслеживание сбоев. Метод наилучшим образом работает, когда группа процессов мала, а членство в группе постоянно и никогда не меняется.
- В распределенном алгоритме все процессы вынуждены участвовать во всех решениях, касающихся входа в критические области. Если один из процессов оказывается неспособным справиться с такой нагрузкой, маловероятно, что возымеет успех попытка их всех сделать то же самое параллельно.
- Алгоритм можно модифицировать так, чтобы разрешить процессу вход в критическую область после того, как он соберет разрешения простого большинства, а не всех остальных процессов.
- Несмотря на все возможные улучшения, этот алгоритм остается более медленным, более сложным, более затратным и менее устойчивым, чем исходный централизованный алгоритм.

Алгоритм маркерного кольца

- Программно создается логическое кольцо, в котором каждому процессу назначается его положение в кольце. При инициализации кольца процесс О получает маркер, или токен (token).
- Маркер циркулирует по кольцу. Он передается от процесса k процессу k+ 1 (это модуль размера кольца) сквозными сообщениями. Когда процесс получает маркер от своего соседа, он проверяет, не нужно ли ему войти в критическую область.
- Если это так, он входит в критическую область, выполняет там всю необходимую работу и покидает область. После выхода он передает маркер дальше.
- Входить в другую критическую область, используя тот же самый маркер, запрещено.
- Если процесс, получив от соседа маркер, не заинтересован во входе в критическую область, он просто передает этот маркер дальше.



Распределенные транзакции

Модель транзакций

- Свойство транзакций «все или ничего» — это лишь одно из характерных свойств транзакции. Говоря более конкретно, транзакции:
  - атомарны (atomic) — для окружающего мира транзакция неделима;
  - непротиворечивы (consistent) — транзакция не нарушает инвариантов системы;
  - изолированы (isolated) — одновременно происходящие транзакции не влияют друг на друга;
  - долговечны (durable) — после завершения транзакции внесенные ею изменения становятся постоянными.
- На эти свойства часто ссылаются по их первым буквам — ACID.

Примитив	Описание
BEGIN_TRANSACTION	Пометить начало транзакции
END_TRANSACTION	Прекратить транзакцию и попытаться завершить ее
COMMIT	Подтвердить транзакцию
ROOLUP	Отменить (откатить) транзакцию

Классификация транзакций

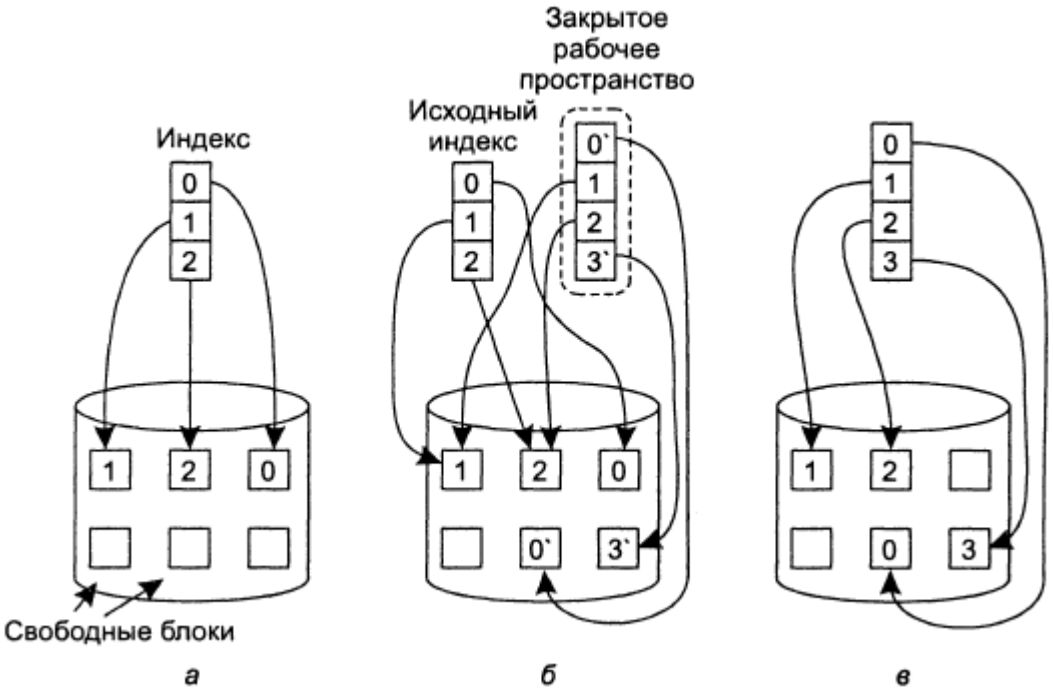
- Плоская транзакция - серия операций, удовлетворяющая свойствам ACID.
- Плоские транзакции имеют одно ограничение - они не могут даватьчастичного результата в случае завершения или прерывания. Другими словами, сила атомарности плоских транзакций является в то же время и их слабостью.
- Вложенные транзакции: Транзакция верхнего уровня может разделяться на дочерние транзакции, работающие параллельно, на различных машинах, для повышения производительности или упрощения программирования.
- Распределенные транзакции: так как вложенные транзакции (плоские) работают с данными, распределенными по нескольким машинам, то такие транзакции известны под названием *распределенных транзакций (distributed transactions).*

Способы реализации транзакций

- Обычно используются два метода:
  - **Закрытое рабочее пространство**.
  - **Журнал с упреждающей записью.**

Закрытое рабочее пространство

- Концептуально, когда процесс начинает транзакцию, он получает закрытое рабочее пространство, содержащее все файлы, к которым он хочет получить доступ.
- Пока транзакция не завершится или не прервется, все операции чтения и записи будут происходить не в файловой системе, а в **закрытом рабочем пространстве**.
- Это утверждение прямо приводит нас к первому методу реализации — созданию для процесса, в момент начала транзакции, закрытого рабочего пространства.



- Индекс файла и дисковые блоки для файла из трех блоков (а).
- Ситуация после того, как транзакция модифицировала блок 0 и добавила блок 3 (б).
- Ситуация после подтверждения транзакции *{в)*

Журнал с упреждающей записью

- Согласно этому методу файлы действительно модифицируются там же, где находятся, но перед тем, как какой-либо блок действительно будет изменен, в журнал заносится запись со сведениями о том, какая транзакция вносит изменения, какой файл и блок изменяются, каковы прежние и новые значения.
- Только после успешной записи в журнал изменения вносятся в файл.

Листинг 5.3. Транзакция

X = 0:

У = 0:

BEGINJRANSACTION:

X = X + 1:

У = У + 2;

X = y * y:

ENDJRANSACTION;

Для каждой из трех инструкций тела транзакции до начала ее выполнения создается запись в журнале, которая содержит прежнее и новое значения, разделенные косой чертой:
- содержимое журнала перед выполнением первой инструкции (x=x+1:):

[x=0/1]

- содержимое журнала перед выполнением второй инструкции (y=y+2:):

[x=0/1]
[y=0/2]

- содержимое журнала перед выполнением третьей инструкции (x=y*y;):

[x=0/1]
[y=0/2]
[x=1/4]

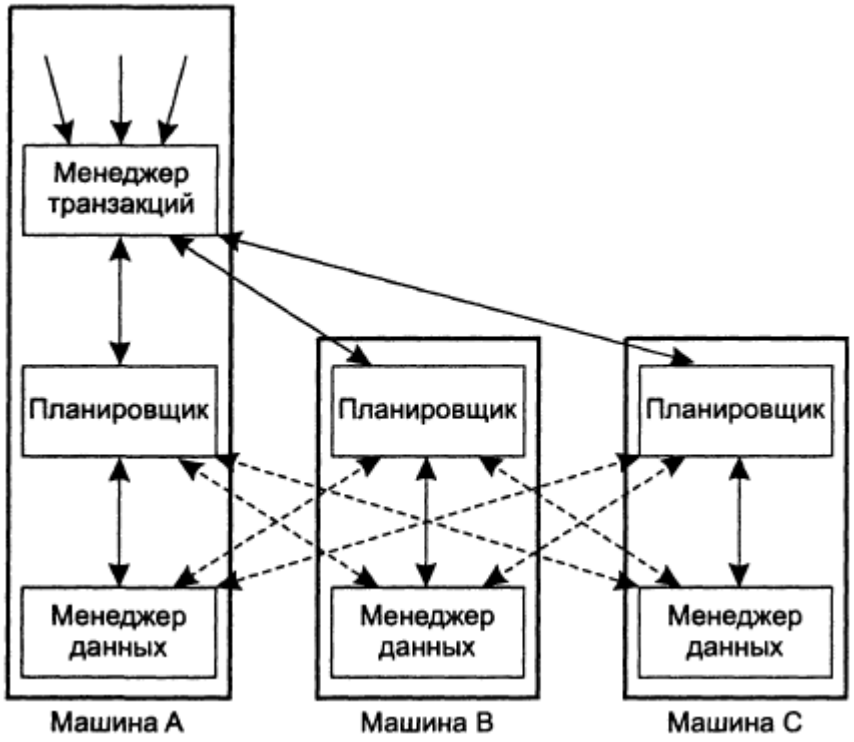
Управление параллельным выполнением транзакций

- Цель управления параллельным выполнением транзакций состоит в том, чтобы позволить нескольким транзакциям выполняться одновременно, но таким образом, чтобы набор обрабатываемых элементов данных (например, файлов или записей базы данных) оставался непротиворечивым.
- Непротиворечивость достигается в результате того, что доступ транзакций к элементам данных организуется в определенном порядке так, чтобы конечный результат был таким же, как и при выполнении всех транзакций последовательно.
- Управление параллельным выполнением лучше всего можно понять в терминах трех менеджеров, организованных по уровням:

На самом верхнем уровне находится *менеджер транзакций.*
На среднем уровне – планировщик.
В самом низу – менеджер данных.

Менеджеры транзакций

- Менеджер транзакций отвечает, прежде всего, за атомарность и долговечность.
- Он обрабатывает примитивы транзакций, преобразуя их в запросы к планировщику.



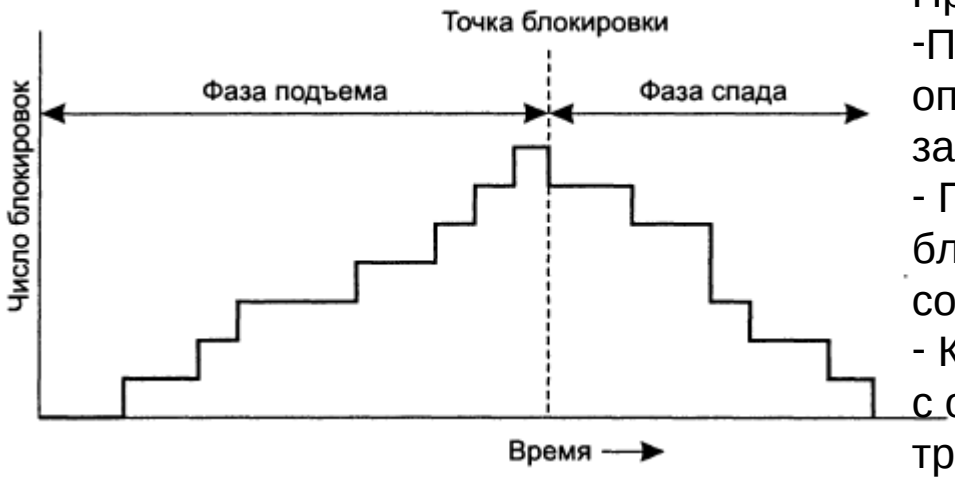
- Каждая машина в этом случае имеет своих планировщика и менеджера данных, которые совместно обеспечивают гарантии непротиворечивости локальных данных.
- Каждая транзакция обрабатывается одним менеджером транзакций. Последний работает с планировщиками отдельных машин.
- В зависимости от алгоритма управления параллельным выполнением транзакций планировщик также может работать с удаленными менеджерами данных.

Изолированность

- Основная задача алгоритмов управления параллельным выполнением — гарантировать возможность одновременного выполнения многочисленных транзакций до тех пор, пока они изолированы друг от друга. Это значит, что итоговый результат их выполнения будет таким же, как если бы эти транзакции выполнялись одна за другой в определенном порядке.

Двухфазная блокировка

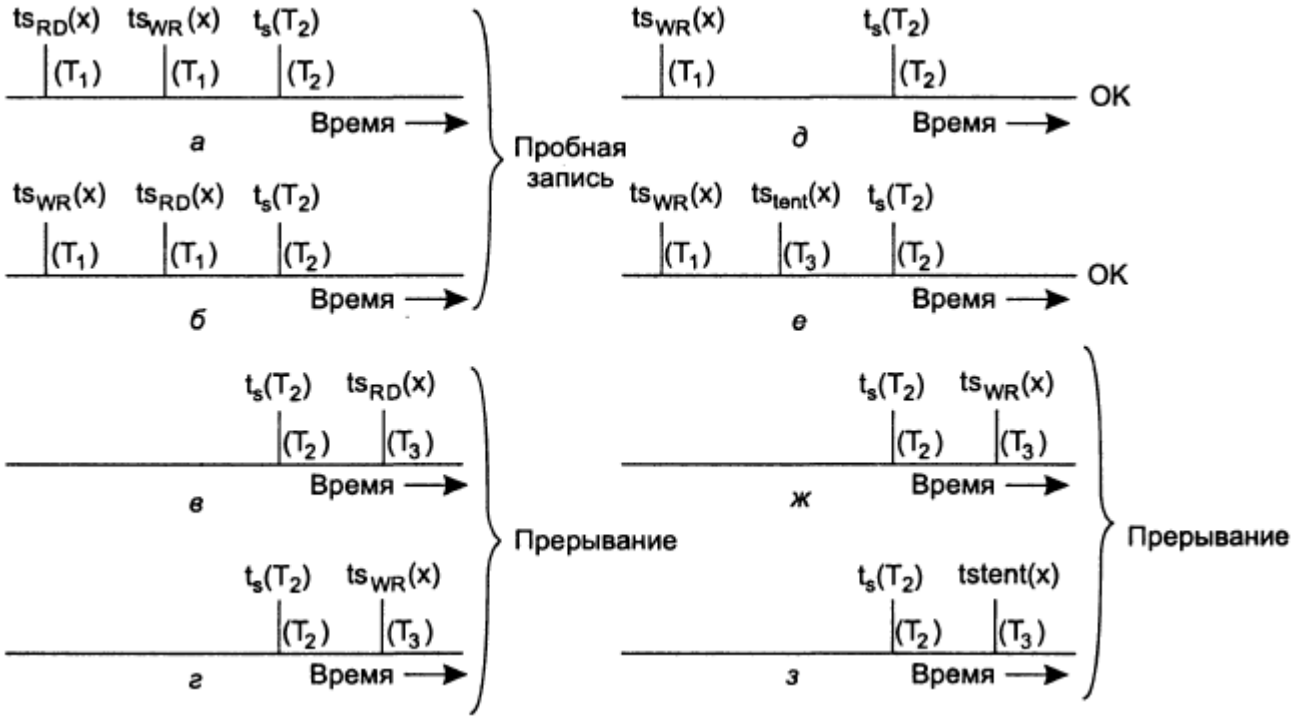
- Самый старый и наиболее широко используемый алгоритм управления параллельным выполнением транзакций — это *блокировка (locking).*
- При *двухфазной блокировке (Two-Phase Locking, 2PL),* планировщик сначала, на *фазе подъема {growingphase), устанавливает* все необходимые блокировки, а затем, на *фазе спада {shrinking phase),* снимает их.



При 2PL выполняются три правила:
-Проверка не конфликтует ли эта операция с другими уже заблокированными операциями.
- Планировщик никогда не снимает блокировку с элемента x, если менеджер сообщает, что он выполняет операцию с x.
- Когда планировщик снимает блокировку с операции установленную по требованию транзакции T, он никогда не делает новую блокировку по требованию этой транзакции.

Доказано, что если все транзакции используют двухфазную блокировку, любой план, сформированный путем перекрытия этих транзакций, сериализуем. В этом причина популярности двухфазной блокировки.

Пессимистическое упорядочение по отметкам времени



Спасибо за внимание !