Синхронизация процессов

Распределенные системы – набор независимых компьютеров, представляющихся их пользователям единой объединенной системой.

(все машины автономны, и пользователи думают, что работают с единой системой)

Плюсы РС:

- они облегчают интеграцию различных приложений, работающих на разных компах, в единую систему

Минусы РС:

- сложное ПО

- Падение производительности

- проблемы с безопасностью

- могут возникнуть конфликтные ситуации:

1. Несколько процессов не способны одновременно получать доступ к совместно используемым ресурсам, но должны позволять друг другу получать эксклюзивный доступ к ресурсам

2. несколько процессов могут нуждаться в соглашении о порядке прохождения сообщения

Синхронизация в РС.

Методы, используемые для однопроцессорных систем, не подходят для РС, т.к. базируются на использовании разделяемой ОП.

Проблема РС состоит в том, что для них не существует понятие единых, совместно используемых часов.

Алгоритмы для синхронизации:

1. Алгоритмы синхронизации часов (Они основаны на обмене показаниями часов, нужно учитывать задержку на отправку и получение сообщение)
2. Алгоритм голосования (Часто требуется, чтобы один из процессов был координатором (зачем?). Если координатора нет, то нужно, чтобы процессы сами выбрали кто будет координатором. Координатор – управляющий процесс)
3. Взаимные исключения (Доступ к совместно используемым ресурсу имеет максимум 1 процесс. Тут нам пригодился координатор, следит за очередью)
4. Алгоритмы транзакции (Транзакция – набор операции с совместно используемыми данными, при этом вся транзакция либо целиком выполнится, либо нет)

Алгоритмы синхронизации часов.

WWV – Синхронизатор (установочник) единого времени. Тик – 100 нс.

Описание алгоритма:

Если нет приемника ни на одной машине, то каждая машина будет отсчитывать каждый свое время, и нашей задачи будет по возможности их синхронизировать.

У каждой машины есть свой таймер (инициализирует прерывание N раз в секунду). Таймер срабатывает -> обработчик прерываний добавляет единицу к програм. часам ->Программные часы сохраняют часы тиков.

1. **Алгоритм Кристиана (**Сервер времени пассивен. Прочие машины периодически запрашивают у него время и он им отвечает**).**

q- разница между временем

Периодически, гарантированно не реже, чем каждые q/2p сек, каждая машина посылает серверу времени сообщение, запрашивая текущее время. (Отвечает максимально быстро текущее время).

C_{utc} – время в ответе.

Есть две проблемы:

Главная – время не течет назад, никогда. (Если часы отправителя спешат, полученное время может оказаться меньше текущего значения у отправителя.). Нельзя просто подставить, т.к. скомпилированные файлы помечены более ранним времени.

Вторая проблема – ответ с сервера требует ненулевого время. (зависит от нагрузки сети).

По Кристиану, надо измерять эту задержку. Отправителю достаточно разницу времени отправки запроса и прихода ответа записать по одним часам. Если известно время обработки прерывания, то оценка может быть улучшена и равна T1 – T0 – I, лучшая оценка времени в одну сторону.

Алгоритм работы:

1. Инициализация. (T3 (по заданию) = T1)
2. Запрос клиентом времени сервера (T0 = T1)
3. Получение времени с сервера:
   А. Считаем время поправки (T1 – T0 – I) / 2
   Б. Расчет поправки времени клиента (Время поправки сервера (предыдущий расчет) + C_{utc}(Время, пришедшее с сервера) – T1 (Текущее время))
   Если отрицательно, то «Приостановить таймер».
   Иначе прибавить поправку.

2. **Алгоритм Беркли (**решена главная проблема алгоритма кристиана, время может идти назад**)**

В данном алгоритм сервер времени (демон времени) активен. Время от времени он опрашивает каждую из машин, какое время на ее часах. По ответам он вычисляет среднее время и предлагает всем установить часы на это время (если отстают) или замедлить часы (если ушли вперед).

Метод применим для систем, не имеющих WWV.

3. **Логические часы (**Синхронизация не обязательно должна быть абсолютной**)**

Если процессы не взаимодействуют, то их не надо синхронизировать (не будет проблем). Важен порядок процессов (запросов).

Алгоритм работы:

1. Процесс отправитель посылается сообщение в текущий момент времени, а получатель получает его на следующем такте (у каждого процесса он свой, величина такта).
2. Время отправителя должено быть меньше (или равно на следующем такте) времени получения. Если нет, то выполнить сихнронизацию. Путем присваивания времени отправителя + 1.

Алгоритмы голосования:

Главное чтобы существовал координатор. Если все процессы одинаковые, то выбрать нельзя.

Алгоритмы голосования пытаются найти процесс с максимальным номером и назначить его координатором.

1. **Алгоритм «забияки»**

а. Когда один из процессов замечает, что координатор больше не отвечает на запросы, то он инициализирует голосование. (посылает всем процессам с номером больше, чем у него, сообщение с типом голосование)

б. Если никто не ответил, то процесс стал координатором. Иначе если ответил процесс с большим номером, то он становится координатором, если несколько, то голосованием продолжается от следующего процесса (среди ответивших).

ОК – процесс готов стать координатором и работает -> В результате все процессы отпадут, кроме одного -> Когда останется один, то победитель отправит всем сообщение о том, что он координатор.

Когда процесс выходит из неработающего состояния, то он организует голосование -> Если он имеет самый большой номер, то он становится координатором.

2. **Кольцевой алгоритм (основан на использовании кольца процессов)**

Предполагается, что каждый процесс знает, кто является его приемником (процессы упорядочены).

Алгоритм работы:

а. когда процесс замечает, что координатор не отвечает, то он посылает сообщение следующему работающему процессу в очереди с указанием номера процесса отправителя в сообщении

б. Если приемник не работает, то процесс указывается как неработающий и сообщение отправляем следующему в очереди. Иначе процесс получает сообщение и пересылает его следующему с добавлением в сообщение отправителя.

в. Когда сообщение сделало круг, то процесс запускает новое сообщение о том, что он координатор.

Взаимные исключения:

Используем критические области для сихнронизации доступа к ресурсу.

Процесс сначала входит в критическую область, и только потом может изменить или считать используемые структуры данных (чтобы убедиться, что ни один из процессов вместе с ним не использует общие структуры данных).

1. **Централизованный алгоритм**

Один из процессов выбирается координатором (процесс с самым большим сетевым адресом).

а. Каждый раз, когда процесс хочет зайти в КС он посылает сообщение координатору номер КС, в которую он хочет войти и запрашивает разрешение войти.

б. Если ни один из процесссов в данный момент не находится в КС, координатор посылает ответ с разрешением на доступ. После получения ответа процесс, запросивший доступ, входит в КС.

Иначе, если процесс уже находится в КС и другой процесс запрашивает доступ к этой КС, то координатор знает, что в КС есть процесс и не дает разрешение на вход и блокирует процесс.

в. Когда процесс выходит из КС то он посылает сообщение координатору, отказываясь от эксклюзивного доступа. Координатор выбирает первый процесс из очереди к КС, разблокирует данный процесс и разрешает вход в КС.

Особенности алгоритма:

1. Позволяет войти в КС только 1 процессу
2. Честный алгоритм (Соблюдается очередь запросов)
3. Никакой процесс не ждет вечно
4. Используется 3 сообщения (запросить, разрешить,)

Минусы: наличие одного неработающего места часто недопустимо, система может перестать работать.

2. **Распределенный алгоритм**

Данный алгоритм требует наличия полной упорядоченности событий в системе. В любой паре событий (например отправка сообщений) должно быть известно, какое событие произошло первым.

Алгоритм работы:

1. Вход в критическую секцию (процесс создает сообщение, содержащее имя критической секции, свой номер и текущее время, после отправляет его всем процессам, включая себя (посылка сообщения надежная – есть подтверждение получения на каждое письмо)) -> вместо сообщений может быть использована доступная надежная групповая связь.
2. Реакция на сообщение:
   а. Если получатель не в КС и не собирается туда, то он посылает ОК

б. Если получатель в КС, то не отвечает и помещает запрос в очередь
в. Если получатель собирается войти в КС, но еще не сделал этого, то сравнивается время отправки сообщения. Если его время меньше, то сообщение в очередь. Если больше, то отвечает ОК.
г. Кто собрал все сообщения тот и заходит в КС.
д. Когда покидает КС, то отсылает сообщение ОК всем процессам в их очереди (хз что тут написано)

Когда приходит запрос, получатель посылает ответ всегда, разрешая или запрещая доступ. Если ответ или запрос утеряны, то выжидается время и либо придет ответ, либо получатель находится в нерабочем состоянии.

3. **Алгоритм маркерного кольца**

Создается логическое кольцо, где каждый процесс имеет положение в кольце. Маркер циркулирует по кольцу (маркер назначается процессу 0 при инициализации).

Алгоритм работы:

a. маркер переходит от процесса К к процессу k+1 с помощью сквозных сообщений.

b. В случае получения маркера от соседа, процесс проверяет, не нужно ли ему войти в КС, если да, то заходит и работает. Если нет, переходит дальше.

Когда работа в КС закончена, то он выходит и передает маркер дальше.

c. Если процессы не хотят войти в КС, то маркер просто циркулирует по кольцу

Минусы: маркер может быть потерян и никто не узнает (может быть просто занят другим процессом)

Распределенные транзакции

Алгоритмы взаимного исключения обеспечивают одновременный доступ не более чем одного процесса к совместно используемым ресурсам (файл, принтер и т.д.). Транзакции тоже защищают общие ресурсы от одновременного доступа к ресурсам. Они превращают процессы доступа и модификации множества элементов данных в одну атомарную операцию. Если процесс решит во время транзакции повернуть назад, то все файлы восстановятся на прежний уровень.

Алгоритм транзакции:

1. Процесс объявляет, что хочет начать транзакции с одним ресурсом
2. Когда инициатор объявляет, что все сделано, идет опрос всех ресурсов на подтверждение этого.
3. Если процессы отказываются, то все восстанавливается как было, иначе все сохраняется.

1. <Закрытое рабочее пространство>
2. Журнал с упреждающей записью

Метод:
Тут модифицируются файлы, а не копии, но перед изменением любого блока производится запись в спец. Файл – журнал регистрации. Указывается вся инфа (какая транзакция, какой файл, какой блок, новое и старое значение изменяемого блока) -> после успешной работы делаются изменения в исходном файле.

Протокол двухфазной фиксации транзакций (Для взаимодействия нескольких процессов на разных машинах), его алгоритм:

1. Координатор начинает транзакцию, делая запись в журнале, посылая всем процессам, учавствующих в транзакции сообщение «Подготовиться к фиксации»
2. После получения сообщения процессы проверяют, готовы ли они к фиксации, делают записи в журнале и посылают координатору «готов к фиксации». Если кто-то не готов, то все откатывается координатором.
3. Если все готовы, то координатор делает запись и посылает команду «фиксировать» всем подчиненным процессам.
4. Процессы выполняют команду, фиксируют изменения и завершают транзакцию.