**11. Системни средства за реално**

**Време**

**Системно време и таймери**

Синхронизацията е необходима при:

комуникации между процесите, подреждане на разпределени събития – право на достъп, бюлетин, транзакции, използване на системното време като аргумент – пример make команда в UNIX. В РС (за разлика от уни- и мултипроцесорите) програмните компоненти може да са разположени на компютри с разлика в системните времена – фиг. 11.3.1 – десинхронизация

(*clock skew*) поради разлика в тактовата честота на осцилаторите и при настройката на системата. Системното време се отчита от таймер – кристален осцилатор + брояч + регистър за броя импулси за 1 сек. – с генерация на системно прекъсване (обикновено с интервал 1 сек.);

системният часовник е процес, който отброява прекъсванията *С* по таймер. За глобална координация се използва UTC – Universal Time, Coordinated – което се

разпространява чрез късовълнови радиостанции от националните институти по стандартизация и геостационарни сателити. Целта е d*C*/d*t* = 1, ∀*t*; реалните осцилатори в масовите компютри работят с относителна грешка ρ ≈ 10-5, т.е. 1-ρ ≤ d*C*/d*t* ≤ 1+ρ, ρ е максимално отклонение (maximum drift rate) с възможно избързване или изоставане – фиг. 11.3.2. Отклонението между два системни часовника за време Δ*t* е δ ≤ 2ρΔ*t*, и ако това е необходимата горна граница на десинхронизация (skewing), се налага ресинхронизиране с период δ/2ρ сек

**Синхронизиращи алгоритми за**

**системното време:** Базират се алтернативно на: времеви сървер, който се синхронизира по UTC, или усреднява системното време на възлите, разпределени схемиза ресинхронизация от тип р2р.

Централизирана (сърверна) синхронизация: алгоритъм на Christian (1989 – ***пасивен сървер с UTC***): периодични заявки от

системните възли към времевия сървер; Проблеми: а)закъснение в цикъла заявка-обслужване-отговор – затова корекцията се прави като към полученото време от сървера се добавая обикновено половината (възможни вариации и по-сложни алгоритми) от закъснението на отговора

(измерено на локалната машина) – фиг. 11.4.1. б)Коригира само избързването (винаги!) – налагат се постепенни корекции при всяка следваща заявка – напр. корекцията с 2ms вместо установените 10ms (независимо от посоката).

Алгоритъм на Berkeley UNIX (1989 – ***активен сървер***, демон): периодична проверка на локалните системни времена във възлите и изравняване към средна стойност (***без връзка с UTC*** предавател) – фиг. 11.4.2.

**р2р синхронизация**

Базира се на периодично общодостъпно предаване на локалното време от всеки възел. След определено изчакване в началото на всеки период, възлите

изчисляват локално време – примерно чрез усредняване с евентуално игнориране на екстремните стойности; Параметри: период на гласуване *R*, период на изчакване *S* << *R* и брой на игнорираните екстремни стойности *m* (алгоритъмът изисква начален синхронен момент за отчитане на периодите *Т*0). протокол за мрежово време (Network Тime Protocol, NTP) – осигурява синхронизация в Интернет с точност до 50 мсек.

**Протокол за тотално подреждане**

Прилага логическа синхронизация с времеви марки за еднакво подреждане на събитията

(получаване на съобщения). При групово предаване (multicasting) – напр. при коригиране на записите в репликирана база данни. При групово предаване на съобщения с времеви марки изпращащия процес като член на групата получава своите съобщения и то в реда на изпращането им и без загуби.

Всеки приемащ процес записва получените съобщения в локален буфер по реда на времевите марки и потвърждава приемането до процесите в групата; потвържденията също се маркират (дистанцирано от съотв. съобщение). Същевременно се прилага и алгоритъма на Лампорт за положителни корекции на локалното логическо време. Всички съобщения – вкл. потвържденията! – са групови (независимо дали са

предназаначени за всички процеси в групата). Локалните буфери са опашки (FCFS) от които съобщенията се предават към съответните локални приложения, като се изтриват от буфера (както и техните потвърждения). Резултат: всички локални буфери са с еднакво подреждане на съобщенията и потока от съобщения към всяко локално приложение е идентичен (N.B.: еднаквото подреждане обаче не гарантира запазване на реда на възникване на събитията в реално астрономическо време ⇒ алгоритъма на Лампорт е приложим за събития, между които няма причинно-следствена връзка.

**Протокол за съхранено подреждане**

Позволява тотално подреждане на събития при запазване на реда им в реално време – напр. при публикуване на дискусионни и новинарски бюлетини, където е важна не само идентична подредба, но и запазване на причинно-следствената връзка – т.е. съхранено подреждане (causally ordering). Прилага векторна маркировка (vector timestamp): Всеки процес Рi поддържа свой вектор от броячи Vi, чиито елементи отразяват броя събития, настъпили в процесите с съответен индекс – Vi[j] = брой настъпили събития в Pj; Vi[i] = брой събития в Pi. За целта когато Pi изпраща съобщението m, към него добавя (т.нар. piggybacking) и текущата стойност на своята вектор Vi като векторна марка vt. По този начин получаващият съобщението m процес Pj e информиран за броя събития, възникнали във всички процеси преди Pi да изпрати m – т.е. общия брой събития, от които изпращането на m може (потенциално) да е следствие. При получаването на m Pj прави корекциите Vj[k] = max{Vj[k], vt[k]} и Vj[i]++, при което Pj вече разполага с броя събития-съобщения, които предхождат (евентуално като причина) m (и съответно – ако има такива – може да ги изчака)

**Представяне на глобалния статус**

***глобалния статус*** се състои от:

***1)локалния статус*** на всеки процес

***2)съобщенията в транзит*** (напуснали локалния изходен буфер на изпращащия процес, но недоставени в локалния входящ буфер на приемащия процес/и).

Локалният статус на процесите е контекстно-зависим – при разпределена БД той може да включва само записите в БД без междинните резултати на обработка; при mark-and-sweep разчистване може да се състои само от маркировката на скелетоните, прокси и обектите от адресното пространство на съответния процес.

Глобалния статус на РС се използва най-често за детекция на мъртва точка (deadlock) или край на разпределената обработка (и в двата случая изпълнението на всички локални процеси е преустановено и няма съобщения в транзит; интерпретацията е предмет на анализ). При РС е важна изискването за *свързаност* (*консистентност*) на глобалния статус – т. нар. ***заснемане на РС*** (***distributed snapshot***) – напр. ако п-с *P* е получил съобщение от *Q*, заснемането трябва да съдържа и запис, че *Q* е изпратил това съобщение

**Алгоритъм за заснемане на глобален**

**статус**

РС се разглежда като набор процеси, свързани с преки симплексни канали (еднопосочни, за разлика от дуплексните и полу-дуплексните) от тип точка-точка (напр. ТСР връзки). Алгоритъмът се инициира от произволен процес *Р* с регистриране на локалния си статус и изпращане на маркер-заявка за заснемане на глобалния статус по всичките си изходящи канали. Процесът *Q* получава заявката по свой входящ канал *С* след което: [заснемане на процес:] регистрира своя локален статус и размножава заявката по своите изходящи

канали; *Q* e наследник, а изпращащия заявката процес е предшественик **ИЛИ**

[заснемане на канал:] ако вече е получил заявката (по друг свой входящ канал) и е регистрирал локалния си статус, той регистрира статуса на канала *С* – т.е. съобщенията, които е получил по този канал в интервала от регистриране на локалния статус до получаване на последния маркер по канала *С.* Краят на заснемането за *Q* е когато получи маркер по всеки свой входящ канал и изпълни горната стъпка; заснетият от него локален статус се изпраща на *P* (възможни варианти с цел рекурсивно описание на процеса). *Р* разполага с глобалния статус на системата когато получи локалните статуси на своите наследници (и рекурсивно–на техните наследници). Няколко заснемания могат да бъдат инициирани така че да протичат едновременно – за целта маркерите съдържат идентификатор на инициатора (който се използва и за изпращане на локалния статус)

**Критични зони с взаимно изключване**

В унипроцесорите критичните зони за взаимно изкл. на достъпа до споделени ресурси се управлява с механизмите на ключалки-семафори и монитори. В РС тези подходи се имплемент. от централизирани алгоритми за управление на достъпа, но се прилагат също и разпределени и резервационни алгоритми. Централизирано взаимно изключване: базира се на излъчен координатор, към който се отправят заявките за достъп до критична зона. Заявките се потвърждават по реда на постъпване. Процесите с непотвърдени заявки изчакват

след освобождаване на критичната зона чакащия (блокиран) заявител получава потвърждение (и достъп.. Ограничен служебен обмен, но ниска отказоустойчивост; в този вариант заявителя не може да различи изчакване от блокирал координатор

**Разпределено взаимно изключване**

Базира се тотално подреждане на събитията с надеждни (потвърдени) групови комуникации. Заявителят изпраща съобщение с името на критичната зона, своя ид. и локалното време. Всеки получател извършва алтернативно следното:

- връща ОК съобщение ако не е или не чака достъп в тази критична зона;

- ако е в критичната зона, не отговаря, а буферира локално заявката

- ако е изпратил собствена заявка за същата критична зона, сравнява двете времеви марки и ако има по-късна (по-голяма) марка, изпраща ОК на заявителя, в противен случай не отговаря, а буферира локално отдалечената заявка.

Заявителят изчаква ОК от всички останли процеси и заема критичната зона. След напускане на критичната зона, процесът изпраща ОК на всички заявители от лок. си опашка за тази зона и ги изтрива от нея.

**Резервирано взаимно изключване Tocken**

**Ring**

базира се на логическо подреждане на п-сите в пръстен;

Стартиращия процес освобождава съобщ. Token. Служебното съобщение се предава последователно между процесите, давайки право на текущия процес на достъп до

критичната зона, след излизане от която съобщението-token се предава към следващия процес в пръстена. Получаването на token дава права на еднократен достъп в една от критичните зони. При загубен token възстановяването е контекстнозависимо, тъй като е базирано на времеинтервали

**Разпределени транзакции**

• транзакциите са механизъм за синхрониз. на съвместната работа на устройствата в системата (първоначално при унипроцесори-те), на взаимодействащи процеси и др.

• функционират на ***принципа “всичко-или-нищо”***: или се изпълняват докрай, или процесите се връщат в състоянието преди началото на изпълнение на транзакцията (примери: обслужване с банкомат, електронна търговия, он-лайн резервации)

• синхронизацията с транзакции се базира на специални *примитиви*, които се поддържат от ОС или се интерпретират като езиково разширение – т.е. обръщения към системата, библиотечни процедури или езикови изрази

(специализирани, но в тялото на транзакцията може да присъстват и изрази с общо предназанчение)

• наборът транзакционни примитиви е контекстноориентиран, но за синхронизация на обслужването винаги включва begin\_transaction, end\_transaction, abort\_transaction и евентулно read и write – фиг. 11.16.

**Свойства на транзакциите (ACID), блокови**

**транзакции**

атомарност (Atomic) – т.е. прозрачност – резултата от транзакцията е или като от

еднократна моментална операция или операция, изобщо отсъства все едно не е правен опит да се изпълни (“all-or-nothing”) – напр. транзактно добавяне на байтове към файл преди края на транзакцията файла е достъпен само в началния си вид (без междинни състояния)

логичност (Consistent) – съхраняване на системните константи – примера с банковия

трансфер със запазване на общата сума пари – по време на изпълнение на самата транз. принципа се нарушава, но друг п-с няма достъп до манипулираната информация, така че нарушението е прозрачно

изолираност (Isolated | serializable) – конкуретните (едновременни) транзакции се изпълняват като последователни съгл. определени принципи на подреждане

устойчивост (Durable) – след изпълнението на транзакцията резултатите от нея не могат да се отменят ACID- | flat- (т.е. блокови) транзакциите не допускат съхраняване и достъп до междинни резултати, което не винаги е желателно, напр. резервацията на серия полети

**Разпределени транзакции**

при тях декомпозицията на супертранзакц. в

субтранзакции не следва логическото разделение, а се определя от структурата на разпределения контекст – напр. разпределна база данни, върху всеки от дяловете на която оперира отделна субтранзакция. Пример: междубанков трансфер със субтранзакции върху различни бази данни – фиг. 11.19. Контраст с блоковите транзакции: блокова е напр. транзакция за начисляване на лихва по сметка (в една база данни)

**Имплементация на транзакциите**

С резервирано работно пространство или с дневник (log-файл). Резервираното работно пространство изисква при стартирането на транзакцията целият контекст заедно с входно-изходните файлове се разполага в резервирано (private) работно пространство; операциите не се регистрират във файловата система до приключването й.

За оптимизиране, в работното пространство се копират само съответните блокове от файловете, отваряни за четене – както и системния индекс на съответния файл. Обработата се извършва върху копието на блоковете и индекса; след приключване на транзакцията, индекса и блоковете се коригират и във файловата система – фиг. 11.20. При метода с log-файл всеки от записите на транзакцията се извършва направо върху блоковете на файловата система, но предварително се регистрира с индекс на блока, старо и ново съдържание (writeahead log). В случай че транзакцията бъде отменена, регистрационният (log-) файл се използва за възстановяване в обратен ред на записите (LIFO) – “rollback”. Тeзи методи са приложими и за разпределените транзакции, тъй като субтранзакциите оперират локално

**Конкурентно изпълнение на транзакциите**

Конкурентното (едновременно) изпълнение на няколко транзакции изисква контрол на достъпа до техния конткст – напр. файлове и БД-записи – така че резултата да е консистентен т.е. такъв като при последов. им изпълнение. За целта управлението на транзакциите се разслоява йерархично на 3 нива: мениджър транзакции МТ – транслира примитивите на отделните транзакции в заявки за следващото диспеческо ниво (напр. с ид. на транзакцията и [отдалечен] адрес на данните + управляваща информация); диспечер Д – планира реда и момента за извършване на отделните операции от различните транзакции съгласно планиращ алгоритъм (по методите с ключалки и времеви марки); мениджър данни МД – изпъллнява четене и запис в устойчивите структури данни прозрачно за планирането на транзакциите.

**Серийно планиране на конкурентни**

**транзакции**

Серийното планиране запазва резултата от конкурентните транзакции такъв, какъвто би бил при последователното им изпълнение. Пример – фиг. 11.22. – с две коректни и едно некоректно планиране. Коректното планиране разрешава конфликтните операции. Конфликтни операции са тези, които две (или повече) конкуретни транзакции извършват върху общи данни и поне една от тези операции е запис: четене-запис конфликт,

запис-запис конфликт; Конфликтът се разрешава чрез заключване на данните или чрез подреждане с времеви марки. Прилагат се два планиращи подхода:

***песимистичен подход***: операциите се синхронизират ***преди*** изпълнението им т.е. проверяват се за конфликт и ако да – се подреждат преди да бъдат изпълнени

***оптимистичен подход***: операциите се синхронизират ***след*** изпълнението им т.е.

изпълняват се целите транзакции и ако накрая се установи че е имало конфликтни операции, поне една от транзакциите се отменя (абортира)

**Песимистично планиране с двуфазно**

**Заключване:**

**Т**ъй като транзакциите са конкурентни, заявките за заключване подлежат на потвърждение (от Д в зависимост от изискванията на безконфликтното серийно планиране). При двуфазното заключване (two-phase locking, 2PL) заключването се разделя на две фази: нарастване (growing phase): процесите на транзакциите заявяват заключване на съответните данни (чрез заявка от съотв. МТ до Д); заключване е необходимо и при четене;

свиване (shrinking phase): процесите на транзакциите заявяват отключване на съответните данни чрез заявка от съотв. МД до Д – фиг. 11.23.

Важат следните правила за диспечеризация на конкурентните заявки:

- при заявка за операция, Д проверява конфликтността с вече потвърдените заявки и потвърждава заключването или отлага заявката както и изпълнението на заявяващата транзакция (песимистично планиране)

- Д освобождава заключване само след като получи потвърждение от ДМ, че операцията е завършила. - След освобождаване на заключване по заявка на даден МТ (и респ. транзакция), Д не допуска нова заявка от същата транзакция – независимо дали е за същия или друг обект; нови заключвания се допускат преди да е освободено първото от тях; противното е програмна грешка, която отменя самата транзакция

**Песимистично планиране с времеви марки**

при този метод се маркират както заявките, така и данните

заявките се макират с времева марка *s* за началото на съответната транзакция *T* като се прилага алгоритъма на Лампорт за уникалност на марките – т.е. *s*(*T*)

обектите данни *х* се маркират с марки за четене И запис – съотв. *sw*(*x*) и *sr*(*x*) – съответставащи на транзакционните марки *s*(*Tm*) и *s*(*Tn*) на процесите, които последни са извършили съответните операции при конфликт на две заявки се потвърждава тази с по-малка марка (по-ранно стартиране)

при заявка read(*T*, *x*): *s*(*T*) < *sw*(*x*) → *Т* се отменя (абортира) – *х* е променян след старта на *Т* при заявка read(*T*, *x*): *s*(*T*) > *sw*(*x*) → заявката на *Т* се потвърждава, като *sr*(*x*) = max{*s*(*T*), *sr*(*x*)}при заявка write(*T*, *x*): *s*(*T*) < *sr*(*x*) →*Т* се отменя (абортира) – *х* е прочетен след старта на *Т*

при заявка write(*T*, *x*): *s*(*T*) > *sr*(*x*) → заявката на *Т* се потвърждава, като *sw*(*x*) = max{*s*(*T*), *sw*(*x*)} примери – фиг. 11.25 планирането с времеви марки води по-често до отмяна на транзакции от това със заключване,

защото отменя транзакции, които при заключването само биха били отложени; същевремнно при времевото маркиране не възниква мъртва точка (поради уникалността и маркиравката на данните)

**Оптимистично планиране с времеви марки**

- конкурентните транзакции се изпълняват докрай без заключване и сравняване на времеви марки, като същевременно се регистрират всички обекти данни, д , върху които е изпълнено четене или запис

- в края на транзакцията се проверява дали нейните операции са консистентни на операциите на останалите конкурентни транзакци и при откриване на промяна в даден обект след стартирането на тази

транзакция, тя се отменя (аналогия с песимистичното времево планиране)

- това планиране се имплементира с резервирано работно пространство за всяка транзакция, чието съдържание се записва във файловата система само при успешно изпълнение на транзакцията

- особености на оптимистичното планиране: висок паралелизъм – няма отлагане и мъртви точки – но при отмяна на транзакция, тя се рестартира отначало при високо натоварване на РС (ρ >80%) производителността е по-лоша от тази на песимистичното планиране рядко се прилага за РС и понеже се възприема като по-сложно за имплементация