

# 11. Системни средства за реално време

Васил Георгиев



[ci.fmi.uni-sofia.bg/](http://ci.fmi.uni-sofia.bg/)



[v.georgiev@fmi.uni-sofia.bg](mailto:v.georgiev@fmi.uni-sofia.bg)

# Съдържание

- Синхронизация и системно време
- Протоколи за подреждане
- Глобален статус
- Взаимно изключване
- Разпределени транзакции

# 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  $C$  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  $dC/dt = 1, \forall t$ ; реалните осцилатори в масовите компютри работят с относителна грешка  $\rho \approx 10^{-5}$ , т.е.

$$1-\rho \leq dC/dt \leq 1+\rho,$$

$\rho$  е максимално отклонение (maximum drift rate) с възможно избързване или изоставане – фиг. 11.3.2

- ✦ отклонението между два системни часовника за време  $\Delta t$  е

$$\delta \leq 2\rho\Delta t,$$

и ако това е необходимата горна граница на десинхронизация (skewing), се налага ресинхронизиране с период  $\delta/2\rho$  сек.

# Синхронизиращи алгоритми за системното време

- ✦ базират се алтернативно на:

- ✦ времеви **сървер**, който се синхронизира по UTC, или усреднява системното време на възлите
- ✦ **разпределени схеми** за ресинхронизация от тип p2p

- ✦ централизирана (сърверна) синхронизация:

- ✦ алгоритъм на Christian (1989 – **пасивен сървер с UTC**): периодични заявки от системните възли към времевия сървер; проблеми:
  - ✦ закъснение в цикъла заявка-обслужване-отговор – затова корекцията се прави като към полученото време от сървера се добавя обикновено половината (възможни вариации и по-сложни алгоритми) от закъснението на отговора (измерено на локалната машина) – фиг. 11.4.1.
  - ✦ 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  $S \ll R$  и брой на игнорираните екстремни стойности  $m$  (алгоритъмът изисква начален синхронен момент за отчитане на периодите  $T_0$ )
- ✦ протокол за мрежово време ([Network Time Protocol](#), NTP; Mills, 1992) – осигурява синхронизация в Интернет с точност до 50 мсек.

# Синхронизация за подреждане

- прилага се за подреждане на събития, когато
  - не е важно съответствието между машинното и физическото време – логически часовник
  - не е необходима синхронизация на машинното време между възлите, а само еднакво подреждане на отдалечени събития
- базира се на
  - релация за предходност (happens-before rel.) между събитията:  $a \rightarrow b$  ( $a$  предхожда  $b$ ), транзитивна
  - релация за конкурентност – когато не може да се определи реда на [две] отдалечени събития – напр. в два асинхронни процеса
  - логическо време  $C$ :  $a \rightarrow b \Leftrightarrow C(a) < C(b)$ ; то се коригира само в посока нарастване
- в РС логическото време е локално за всеки възел

# Синхронизация с времеви марки (timestamps)

- (Lamport, 1978): синхронизиращи съобщения между възлите с времеви марки на локалните логически времена
  - ако получаващия процес има по-малка стойност на локалното логическо време от марката на изпратеното съобщение, той коригира своя логически часовник (само в положителна посока!) към стойност (марка+1) – фиг. 11.7.
  - изискване: няма две събития с еднакво  $C$  – ако синхронизацията процес изпраща/приема едновременно две съобщения с времеви марки, той ги дистанцира логически на 1 такт
- допълнително прецизиране на логическото време (за уникални марки) се постига като към целочислената марка се добави процесния идентификатор (или негова производна) като дробна част

# 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 предназначени 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  $\Rightarrow$  алгоритъма на Лампорт е приложим за събития, между които няма причинно-следствена връзка – causality)



# Протокол за съхранено подреждане

- позволява тотално подреждане на събития при запазване на реда им в реално време – напр. при публикуване на дискуссионни 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  $P_i$  поддържа свой вектор от броячи  $V_i$ , чиито елементи отразяват броя събития, настъпили в процесите с съответен индекс –  $V_i[j]$  = брой настъпили събития в  $P_j$ ;  $V_i[i]$  = брой събития в  $P_i$
  - за целта когато  $P_i$  изпраща съобщението  $m$ , към него добавя (т.нар. piggybacking) и текущата стойност на своята вектор  $V_i$  като векторна марка  $vt$
  - по този начин получаващият съобщението  $m$  процес  $P_j$  е информиран за броя събития, възникнали във всички процеси преди  $P_i$  да изпрати  $m$  – т.е. общия брой събития, от които изпращането на  $m$  може (потенциално) да е следствие
  - при получаването на  $m$   $P_j$  прави корекциите  $V_j[k] = \max\{V_j[k], vt[k]\}$  и  $V_j[i]++$ , при което  $P_j$  вече разполага с броя събития-съобщения, които предхождат (евентуално като причина)  $m$  (и съответно – ако има такива – може да ги изчака)

# Примерен сценарий за съхранено подреждане

- ✦ електронен бюлетин с участие на процесите  $P_i$ ,  $P_j$ ,  $P_k$  (и други):
  - ✦  $P_i$  – публикува [в групата] статия (съобщение)  $a$ ; при груповото предаване  $P_i$  добавя към  $a$  и векторната марка  $\mathbf{vt}(a)=V_i$
  - ✦  $P_j$  – публикува пасивно  $a$  след което публикува [в групата] реакцията (съобщение)  $r$ ; при получаването на  $a$   $P_j$  коригира  $V_j$  така че  $V_j[i] > \mathbf{vt}(a)[i]=V_i$ ; при изпращането на реакцията  $P_j$  добавя към  $r$  векторната марка  $\mathbf{vt}(r)=V_j$ ; (подреждането на събитията се регистрира чрез отношението  $\mathbf{vt}(r)[i] > \mathbf{vt}(a)[i]$ )
  - ✦  $P_k$  – публикува пасивно  $a$  и  $r$ ;  $P_k$  получава двете съобщения (незадължително в коректна последователност) но публикува  $r$  само след като:
    - ✦  $\mathbf{vt}(r)[j] = V_k[j]+1$  (т.е.  $r$  е точно следващото съобщение, което  $P_k$  очаква от  $P_j$ ) и
    - ✦  $\mathbf{vt}(r)[i] \leq V_k[i]+1, \forall i \neq j$  (т.е.  $P_j$  не е получил съобщения, които  $P_k$  не е получил към момента на изпращане на  $r$ ; в конкретния пример това е важно само за съобщението  $a$ )

# Представяне на глобалния статус

- **глобалния статус** се състои от
  - **локалния статус** на всеки процес
  - **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
- консистентността на заснемането се представя с разрез (cut) – фиг. 11.11.



# 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. TCP връзки)
- алгоритмът се инициира от произволен процес  $P$  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  $C$  след което
  - [заснемане на процес:] регистрира своя локален статус и размножава заявката по своите изходящи канали;  $Q$  е 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  $C$  – т.е. съобщенията, които е получил по този канал в интервала от регистриране на локалния статус до получаване на последния маркер по канала  $C$
- края 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)
- $P$  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 изключване на достъпа до споделени ресурси се управлява с механизмите на ключалки-семафори и монитори
- ✦ в РС тези подходи се имплементират 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) – фиг. 11.13.
  - ✦ ограничен служебен обмен, но ниска отказоустойчивост; в този вариант заявителя не може да различи изчакване от блокирал координатор

# Разпределено взаимно изключване (Ricart, Argawala - 1981)

- 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 останали процеси и заема критичната зона
- след напускане на критичната зона, процесът изпраща ОК на всички заявители от локалната си опашка за тази зона и ги изтрива от нея
- пример – фиг. 11.14.

# Резервирано взаимно изключване Token Ring

- ✦ базира се на логическо подреждане на п-сите в пръстен; стартирация процес освобождава съобщението 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
- ✦ сравнение между централизираните, разпределените и резервационните алгоритми за взаимно изключване – фиг. 11.15.

# Разпределени транзакции

- транзакциите са механизъм за синхронизация на съвместната работа на устройствата в системата (първоначално при унипроцесорите), 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 предназначение)
- 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 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 транзакция принципа се нарушава, но друг п-с няма достъп до манипулираната информация, така че нарушението е прозрачно
- ✦ изолираност (Isolated | serializable) – конкуретните (едновременно) 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

# Вложени транзакции

- вложени (nested) транзакции – представляват йерархичен дървовиден набор от субтранзакции, първата от които инициира няколко от следващото ниво и т.н. – в съответствие с логическото и каузално (причинно-следствено) разделение на цялата “супертранзакция”; всяка от субтранзакциите е логически независима от изпълнението на останалите (примера с последователните полети – фиг. 11.18.)
- целта е да се постигне ускорено изпълнение при паралелно изпълнение от няколко сървера, но може да се ползват и за съхраняване на междинни резултати
- наборът субтранзакции се счита за изпълнен, само ако главната субтранзакция е изпълнена, а ако не е – заличават се и резултатите на успешно изпълнените дъщерни субтранзакции (което може да породя проблем особено при изпълнение в РС)
- изпълнението на вложените транзакции е рекурсивно: когато главната субтранзакция е изпълнена, за изпълнени се считат и другите завършили субтранзакции по йерархията; резултатите от неизпълнените субтранзакции се заличават

# Разпределени транзакции

- при тях декомпозицията на супертранзакцията 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 ѝ
- за оптимизиране, в работното пространство се копират само съответните блокове от файловете, отваряни за четене – както и системния индекс на съответния файл
- обработката 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”
- тези 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 контекст – напр. файлове и БД-записи – така че резултатът да е консистентен т.е. такъв като при последователното 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 диспечерско 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 – изпълнява четене и запис в устойчивите структури данни прозрачно за планирането на транзакциите
- конкурентно изпълнение в РС (фиг. 11.21.):
  - във всеки възел се стартира двойка от процесите Д и МД, а за всяка транзакция – отделен МТ
  - МТ изпраща генерираните заявки към съответния Д
  - Д може да изпрати планираните от него заявки и към отдалечени МД

# 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) конкурентни 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

- тъй 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



# Варианти на 2PL

- **строго** (strict) 2PL, при което всички заключвания на транзакцията се освобождават след приключване на последното от тях (дори и когато съотв. транзакция завършва с отмяна); така се избягва възможността от каскадни отмени на транзакции, която възниква ако са били обработени резултати от отменени впоследствие транзакции (достъпни са само резултати на вече изпълнени транзакции)
- блокировка в мъртва точка (deadlock - ) при [strict] 2PL настъпва ако две транзакции заявят едновременно две заключвания но в обратен ред
- за избягване на мъртва точка се прилага:
  - служебно подреждане на заявките
  - времеинтервал за откриване на мъртва точка – когато заключването продължи в рамките на този интервал
  - граф на процесите и заключванията за откриване на цикли
- **централизирано** 2PL, при което заявките се обработват от централизиран Д, а достъпът на МТ до МД е разпределен; вариант: няколко Д си разпределят контрола за достъп до данните (primary 2PL)
- **разпределено** 2PL: всеки Д планира достъпа само до локалните данни, но ако данните са репликирани, съответния Д размножава заявката до възлите с реплики



# 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  $x$  се маркират с марки за четене И запис – съотв.  $sw(x)$  и  $sr(x)$  – съответстващи 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  $read(T, x)$ :  $s(T) < sw(x) \rightarrow T$  се отменя (абортира) –  $x$  е променен след старта на  $T$
- при заявка  $read(T, x)$ :  $s(T) > sw(x) \rightarrow$  заявката на  $T$  се потвърждава, като  $sr(x) = \max\{s(T), sr(x)\}$
- при заявка  $write(T, x)$ :  $s(T) < sr(x) \rightarrow T$  се отменя (абортира) –  $x$  е прочетен след старта на  $T$
- при заявка  $write(T, x)$ :  $s(T) > sr(x) \rightarrow$  заявката на  $T$  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; същевременно при времевото маркиране не възниква мъртва точка (поради уникалността и маркиравката на данните)
- варианти: консервативно планиране с времеви марки [Jim Gray, Andreas Reuter: Transaction Processing: Concepts and Techniques. Morgan Kaufmann 1993.] и многовариантно планиране с времеви марки [Ozsu and P. Valduriez. Principles of Distributed Database Systems. Prentice Hall, 1999.]

# Оптимистично планиране с времеви марки

- ✦ конкурентните транзакции се изпълняват докрай без заключване и сравняване на времеви марки, като същевременно се регистрират всички обекти данни, върху които е изпълнено четене или запис
- ✦ в края на транзакцията се проверява дали нейните операции са консистентни на операциите на останалите конкурентни транзакции 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 ( $\rho > 80\%$ ) производителността е по-лоша от тази на песимистичното планиране
  - ✦ рядко се прилага за РС и понеже се възприема като по-сложно за имплементация