Sisteme distribuite – Teorie
9. Impas

Clasificare

- Impasurile in SD sunt similare cu impasurile din sistemele mono-procesor, insa mia complicate
  - Sunt greu de evitat, prevenit sau chiar detectat, si mai greu de tratat la depanare deoarece informatia relevanta este imprastiata pe mia multe masini
  - In sisteme de baze de date efectele sunt extrem de serioase
- Uneori se face distinctia intre doua tipuri de impasuri distribuite:

1. *Impasuri la comunicare:*
   - Apar, de exemplu, cand procesul A incearca sa trimita un mesaj procesului B, care la randul sau incearca sa trimita un mesaj lui C, care incearca sa trimita un mesaj lui A.
   - Exista o serie de scenarii care pot conduce la acest impas, de exemplu lipsa zonelor tampon
2. *Impasuri la resurse:*
   - Apar cand procesele se lupta pentru acces exclusiv la dispozitivele de I/O, fisiere sau alete resurse

Se considera numai impasul la resurse

- Canalele de comunicare, zonele tampon etc sunt de asemenea resurse care pot fi modelate ca resurse
  - Procesele le pot cere si elibera
- Sabloanele de comunicare circulara de tipul descris anterior sunt rare in majoritatea sitemelor:
  - Exemplu: sistemele client-server,
    - Un client poate trimite un mesaj (sau efectua un RPC) la un server de fisiere, care poate trimite un mesaj la un server de disk
    - Este putin probabil ca serverul de disk, comportandu-se ca si client, sa trimita un mesaj la clientul original, asteptand sa se comporte ca un server
  - Conditia de asteptare circulara este improbabil sa se produca ca rezultat numia a unei comunicari

Strategiile care sunt utilizate pentru tratarea impasurilor

1. Algoritmului strutului (ignorarea problemei):
   - popular in SDuri ca si in sistemele mono-procesor
   - Nu exista un mecanism de impas in intregul sistem utilizat pentru programare, control a proceselor, automatizarea gen office si alte aplicatii
2. *Detectare* (permite aparitia impasurilor, le detecteaza, incearca recuperarea).
   - De asemenea popular, deoarece prevenirea si evitarea sunt dificile
3. *Prevenirea* (asigura static faptul ca structural impactul este imposibil)
   - Mult mai dificil decat in cazul mono-procesor
4. Evitarea (evita impasurile prin alocarea cu grija a resurselor)
   - Neutilizat in SD
   - Problema este aceea ca algoritmii propusi trebuie sa cunoasca (in avans) cat din fiecare resursa ii este necesara fiecarui proces –asemenea informatie este rara, si de obicei nedisponibila

Detectarea impasului distribuit

- Cand un impas este detectat intr-un sisteme de operare conventional, o modalitate de rezolvare este aceea de a omori unul sau mai multe procese
  - Facnad acest lucru invariabil, utilizatorii nu vor fi satisfacuti
- Cand este detectat un impas intr-un sistem bazat pe tranzactii atomice, este rezolvat prin abort al unei sau mai multor tranzactii
  - Tranzactiile sunt construite a.i. sa suporte abortarea
  - Cand o tranzactie este abortata deoarece contribuie la un impas,
    - sistemul este restaurat in starea dinaintea tranzactiei,
    - dupa care trazactia poate porni din nou
    - Este probabil ca sa fie cu succes a doua oara

  -> Diferenta este aceea ca, consecintele omoririi unui proces sunt mult mai putin severe cand sunt utilizate tranzactiile fata de cazul cand nu sunt

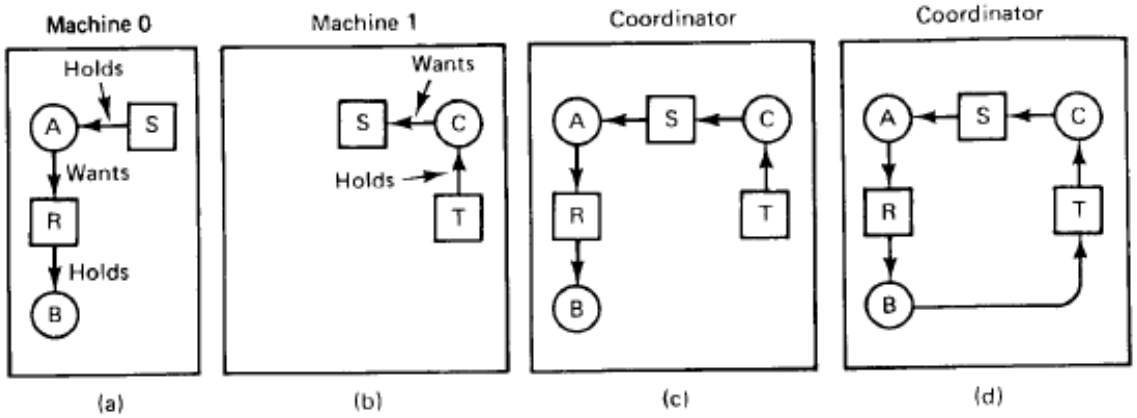
Detectarea centralizata a impasului

- Incearca sa imite un algorit ne-distribuit
- Fiecare masina mentine un graf al resurselor pentru procesele si resursele sale
- Un coordonator central mentine un graf al resurselor din intregul sistem (reunirea grafurilor individuale).
- Variante:
  1. Cand un arc este adugat sau sters din graful resurselor, este expediat un mesaj la coordonator despre actualizare
  2. Periodic, fiecare proces poate expedia o lista de arce adaugate si sterse de la ultima actualizare (necesita mai putine mesajul decat varianta anterioare)
  3. Coordonatorul poate cere informatia de care are nevoie
- Cand coordontorul detecteaza un ciclu, omoara unul dintre procese pentru a intrerupe impasul

Impas fals

- Se considera un sistem cu procesele A si B ruland pe masina 0, si procesul C pe masina 1.
- Exista trei resurse: R, S si T.
- Initial:
  - A detine S si doreste R, pe care nu-l poate avea deoarece B il utilizeaza;
  - C are T si vrea S, de asemenea.
  - Imaginea pe care coordonatorul o are este prezentata in (c)
  - Aceasta configuratie este sigura: odata ce B termina, A primeste R si termina, eliberand S pt.C
- Dupa un timp:
  - B elibereaza R si cere T, o schimbare legala si sigura
  - Masina 0 trimite un mesaj la coordonator anutand eliberarea lui R,
  - Masina 1 expediaza un mesaj la coordonator anuntand faptul ca B asteapta sa primeasca T.
  - Din ppacate, mesajul de la masina 1 ajunge primul, ceea ce duce la construirea grafului (d) la coordonator
  - Coordonator conclude incorect ca exista un impas si omoara un proces

  Asemenea situatie este numita *impact fals*.

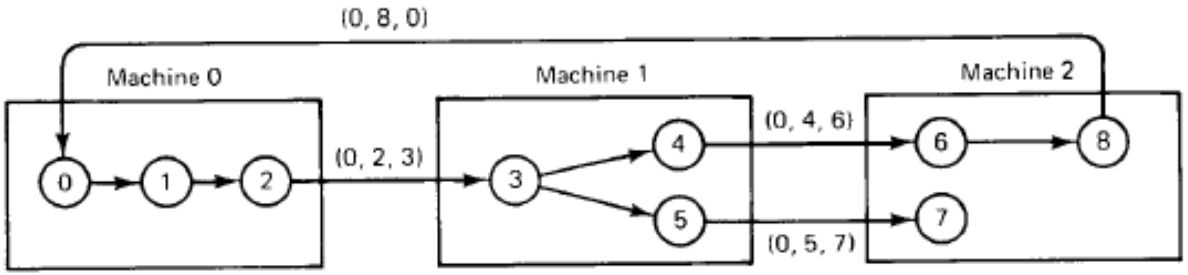


Cale de iesire: utilizarea alg.lui Lamport

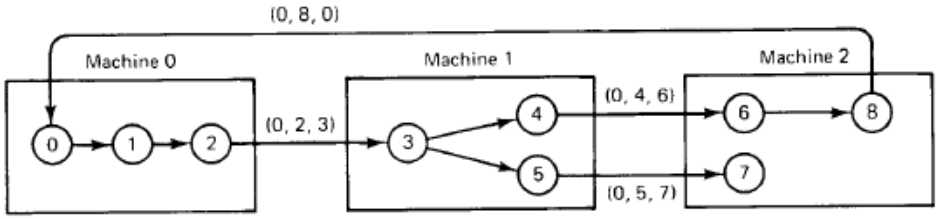
- Deoarece mesajul de la masina 1 catre coordonator este realizata la cererea masinii 0,
  mesajul de la masina 1 catre coordonator va avea o marca de timp ulterioara mesajului de la masina 0 catre coordonator.
- Cand coordonator primeste mesajul de la masina 1 care-l face sa suspecteze un impas,
  expediaza un mesaj catre toate masinile din sistem spunand:
  "Am primit un mesaj cu marac de timp T care conduce la un impas.
  Daca oricare are un mesaj pentru mine cu o marca de timp mai timpurie, trimite-mi-l imediat."
- Cand fiecare masina a masina a replicat, pozitiv sau negativ,
  coordonatorul va vedea arcul de la R la B diaparand,
  astfel incat sistemul este inca sigur
- Desi aceasta metoda elimina impasul fals,
  necesita mentinerea timpului global, ceea ce este costisitor

Detectare distrib. impas: alg. Chandy-Misra-Haas

- Proceselor li se permite sa solicite mesaje multiple la un moment dat
  - Astfel rata de crestere a unei tranzactii poate fi accelerata considerabil
  - Un proces poate asteapta mai mult pentru doua sau trei resurse simultan
- Exemplu:
  - Un graf al resurselor modificat: numai procesele sunt prezentate
  - Fiecare arc trece printr-o resursa
  - Pentru simplicitate resursele au fost omise din figura
  - Procesul 3 pe masina 1 asteapta doua resurse, una detinuta de procesul 4 si alta de procesul 5



Exemplu



- Anumite procese asteapta pentru resurse locale, precum procesul 1, altele, precum procesul 2, asteapta pentru resursele localizate pe o masina diferita
  - Aceste arcuri intre masini fac ca ciclusrile sa pare dificile
- Algoritmul este invocat cand un proces asteaptaa anumite resurse, de exemplu procesul 0 care blocheaza procesul 1:
  - U mesj special proba este generat so trimis la procesul (procesele) care detin respectivele resurse
  - Mesajul consista din trei numere: procesul care tocmai s-a blocat, procesul care expediaza mesajul si procesul la care vrea sa trimita mesjul
  - Mesajul initial de la 1 contine tripletul (0, 0, 1).
  - Cand mesajul soseste, destinatorul verifica daca el insusi asteapta dupa alte procese
  - Daca da, mesajul este actualizat, tinand primul camp dar inlocuidu-l pe cel de-al doilea cu propril sau numar de proces si al treilea cu numarul de proces dupa care asteapta
  - Mesajul este apoi expediat la procesul din cazua caruia este blocat.
  - Daca este blocat de mia multe procese, la fiecare dintre ele este expediat un mesaj (diferit)
  - Mesajele la distanta sunt etichetate (0, 2, 3), (0, 4, 6), (0, 5, 7), si (0, 8, 0).
  - Daca un mesaj trece peste tot si vine inapoi la expeditor, adica se auto-intalneste in primul camp al mesajului, exista un ciclu si sistemul este in impas

Intreruperea impasului din exemplu

1. O modalitate este aceea ca proces care initiaza mesajul proba sa se auto-intrerupa
   - Aceasta metoda are probleme daca mai multe procese invoca algoritmul simultan
     - De exemplu, daca atat procesul 0 cat si 6 se blocheaza in acelasi moment, si ambele initiaza un mesaj proba
     - Fiecare va descopri eventual impasul si fiecare se va auto-intrerupe
2. Un algoritm alternativ este acela in care fiecare proces adauga identitatea sa la sfarsitul unui mesaj proba a.i. este returnat la expeditorul initial, ciclul complet este vizibil
   - Expeditorul poate vedea care procesa are numarul mai mare si cere respectivului sa se intrerupa.
   - In orice caz, daca procese multiple descopera in acelsi timp acelasi ciclu, aleg sa considere aceeasi victima

Prevenirea distribuita a impasului

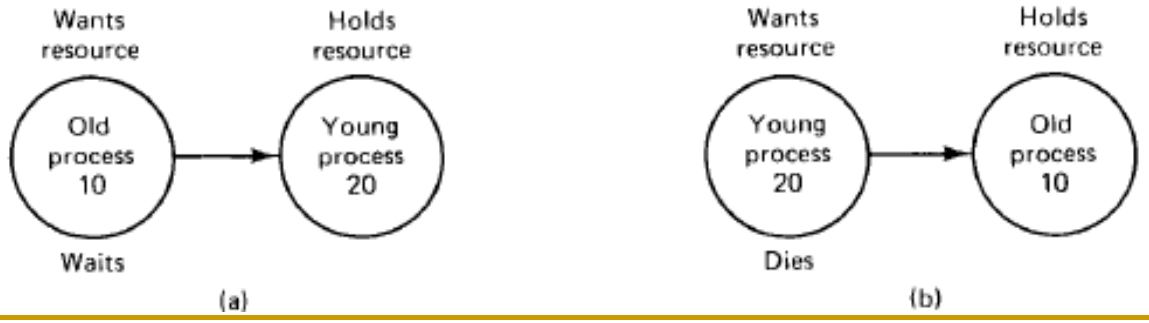
- Consista in designul cu atentie a sistemului a.i. impasurile sunt structural imposibile
- Tehnici variate includ:
  - Permit proceselor sa detina numai o resursa la un moment dat,
  - Solicita proceselor sa ceara toate resursele initial
  - Obligand procesele sa eliberze toate resursee cand solicita una noua
  - Cel mai des: ordonarea tuturor resurselor si solicitatea catre procese sa le primeasca numai in ordine strict crescatoare
    - Un proces nu poate detine o resursa cu numar mare si sa ceara una cu numar mai mic (faceciclurile imposibile)
- Intr-un SD cu timp global si tranzactii atomice, alti doi algoritmi practici sunt posibili:
  - Bazati pe ideea de a asigna fiecarei tranzactii o marca de timp la momentul startarii
  - Esential este ca fiecare doua tranzactii sa nu aiba asignate exact aceeasi marca de timp

Ideile algoritmilor alternativi

- Prima idee :
  - Cand un proces urmeaza sa se blocheze pentru o resursa pe care altul o utilizeaza, o verificare este realizata pentru a vedea care are marca de timp mai mare (adica este mai tanar).
  - Se poate astfel permite asteptarea numai daca procesul in asteptare are o marca de timp mai mica (este mai batran) dect procesul dupa care asteapta
  - In aceasta maniera, urmarind orice lant de procese in asteptare, marcile de timp intotdeauna cresc, a.i. ciclurile sunt imposibile
- Alternativ:
  - Permite proceslor sa astepte numai daca procesul care asteapta are o marca de timp mai mare (este mai tanar) decat procesul dupa care asteapta, in care caz marcile de timp descresc in lant
- Este mai adecvat sa se acorde prioritate proceselor mai batrane
  - Au rulat de mai mult timp, a.i. sistemul a investit mult in acestea si ese probabil sa detina mai multe resurse
  - Un proces tanar care este omorit va imbatrini pana cand va ajunge cel mai batran in sistem, astfel incat aceasta alegere elimina infometarea
- Omorarea unui tranzactii este relativ inofensiva, deoaerce prin definitie poate fi repornita mai tarziu

Exemplu pentru alg. asteapta-mori

- (a): un proces batrin doreste o resursa detininuta de un proces mai tanar.
- (b): un proces tanar doreste o resursa detinuta de un proces mai batran
- Intr-un caz trebuie sa permitem unui proces sa astepte, in celalalt caz sa fie omorat
- Presupunem ca consideram (a) moare si (b) asteapta.
  - Atunci este omorit un proces batran care incearca sa utilizeze o resursa detinuta de un proces mai tanar, ceea ce este ineficient
- Astfel trebuie sa procedam pe dos, asa cum este aratat in figura
- In aceste conditii, sagetile indica intotdeauna directia de crestere a numereor de tranzactie, facand ciclurile imposibile
- Acest algoritm este numit *astepta-mori*.



Remarca

- Presupunand existenta tranzactiilor,
  - este posibil ca resursele sa fie luate de la procesele in rulare
- Cand apare un conflict, in loc s fie ucis procesul care efectueaza cererea, se poate omiri proprietarul resursei
  - Fara tranzactii, omorirea unui proces poate avea consecinte severe, deorece procesul e posibil sa fi modicficat fisiere, de exemplu
  - Cu trnazactii, aceste efecte dispar magic cand tranzactiile mor

Exemplu pentru alg. ranit-asteapta

- O tranzactie este ranita (omarita) si alta asteapta
- Daca un proces mai batrin doreste o resursa detinuta de un proces mai tanar, trnzactia tanarului este omorita
- Procesul mai tanar re-porneste probabil imediat si incearca sa obtina resursa, ceea ce il va determina sa astepte
- Prin contrast cu asteapta-mori in care
  - Daca un batran doreste o resursa detinauta de un tanar, va trebui sa astepte politicos
  - Daca un tanar doreste o resursa detinita de batran, cel tanar este omorit.
  - Va porni din nou si va fi omorat din nou
  - Acest ciclu poate continua de atatea ori cat este necesar procesului batran sa elibere resursa

  Algoritmul ranit-asteapta nu are aceasta proprietate neplacuta