TRƯỜNG ĐẠI HỌC CẦN THƠ
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
KHOA MÔN CÔNG NGHỆ THÔNG TIN

QUẢN TRỊ DỮ LIỆU - CT467

Chương 4: ĐIỀU KHIỂN CẠNH TRANH

Biên soạn:

Ths. Nguyễn Thị Kim Yến

Ntkyen@ctu.edu.vn

NỘI DUNG

1. Kỹ thuật khóa (chốt)
2. Kỹ thuật nhãn (tem) thời gian
3. Giao thức dựa trên tính hợp lệ
4. Quản lý deadlock

2. KỸ THUẬT NHÃN (TEM) THỜI GIAN

Giới thiệu

- Ý tưởng:

  - Đảm bảo các chỉ thị xung đột sẽ được thực hiện theo đúng thứ tự thực thi của các GD.

  - Giao thức này cũng sinh ra được các lịch trình khả tuần tự xung đột và tránh được deadlock

- Chọn một thứ tự thực hiện nào đó cho các giao tác bằng cách gán nhãn thời gian (timestamping)

✓ Mục đích của gán nhãn thời gian: phân biệt được GD nào xãy ra trước, GD nào xảy ra sau trong hệ thống.

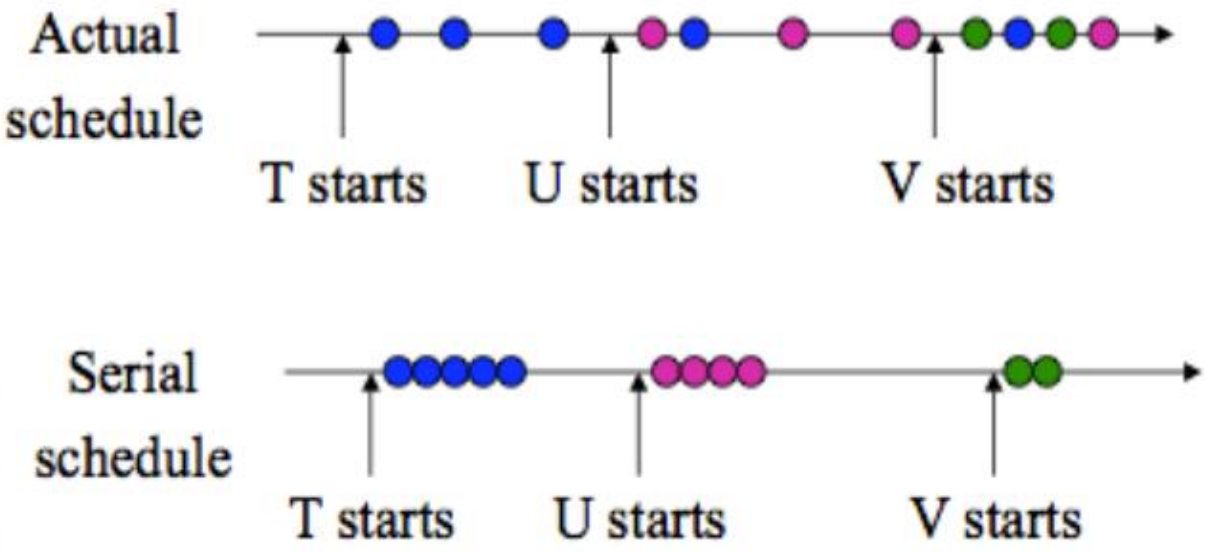
✓ Mỗi GD T_i sẽ được gán một nhãn TG khi nó bắt đầu, ký hiệu: **$TS(T_i)$**

✓ Thứ tự của các nhãn:

- GD T_i bắt đầu trước GD T_j thì $TS(T_i) < TS(T_j)$
- Hoặc, GD T_i bắt đầu trễ hơn GD T_j thì $TS(T_i) > TS(T_j)$

Giới thiệu

- Chiến lược cơ bản:
  - Nếu TS(Ti) < TS(Tj) thì lịch thao tác được phát sinh phải tương đương với lịch biểu tuần tự {Ti, Tj}



Giới thiệu

❖ **Có 2 phương pháp để gán nhãn TG cho các GD:**

- **Đồng hồ máy tính:** Khi GD T_i bắt đầu sẽ lấy giá trị của đồng hồ hệ thống để gán cho TS(T_i).
- **Bộ lặp lịch tự đếm:** Khi GD T_i bắt đầu sẽ được khởi tạo 1 giá trị của biến đếm bất kỳ để gán cho TS(T_i), đồng thời tăng giá trị của biến đếm đó lên.

Nhãn thời gian toàn phần

- Mỗi giao tác T khi phát sinh sẽ được gán 1 nhãn **TS(T)** ghi nhận lại thời điểm phát sinh của T
- Mỗi đơn vị dữ liệu X cũng có **1** nhãn thời **TS(X)**, nhãn này ghi lại TS(T) của giao tác T đã thao tác read/write thành công sau cùng lên X
- Khi đến lượt giao tác T thao tác trên dữ liệu X, so sánh TS(T) và TS(X)
  - **Nếu T muốn đọc X:**
    - Nếu TS(T) ≥ TS(X) thì cho T đọc X và gán TS(X) = TS(T)
    - Ngược lại T bị hủy (abort)
  - **Nếu T muốn ghi X:**
    - Nếu TS(T) ≥ TS(X) thì cho T ghi X và gán TS(X) = TS(T)
    - Ngược lại T bị hủy (abort)

Nhãn thời gian toàn phần

Read(T, X)

```
If TS(X) <= TS(T)
      Read(X);  //cho phép đọc X
      TS(X):= TS(T);
Else
      Abort {T};  //hủy bỏ T
```

Write(T, X)

```
If TS(X) <= TS(T)
      Write(X);  //cho phép ghi
      TS(X):= TS(T);
Else
      Abort {T}; //hủy bỏ T
```

Ví dụ

T_1 $TS(T_1)=100$	T_2 $TS(T_2)=200$	A TS(A)=0	B TS(B)=0	
Read(A)		TS(A)=100		**$TS(A) \le TS(T_1)$: T_1 đọc được A**
	Read(B)		TS(B)=200	**$TS(B) \le TS(T_2)$: T_2 đọc được B**
A=A*2				
Write(A)		TS(A)=100		**$TS(A) \le TS(T_1)$: T_1 ghi lên A được**
	B=B+20			
	Write(B)		TS(B)=200	**$TS(B) \le TS(T_2)$: T_2 ghi lên B được**
Read(B)				**$TS(B) > TS(T_1)$: T_1 không đọc được B**

T1 bị hủy, sau đó khởi động lại T1 với một nhãn thời gian mới

Nhãn thời gian toàn phần

T1 TS(T1)=100	T2 TS(T2)=120	A TS(A)=0
Read(A)		TS(A)=100
	Read(A)	TS(A)=120
	Write(A)	TS(A)=120
Write(A)		

T1 bị huỷ và bắt đầu thực hiện với timestamp mới

T1 TS(T1)=100	T2 TS(T2)=120	A TS(A)=0
Read(A)		TS(A)=100
	Read(A)	TS(A)=120
	Read(A)	TS(A)=120
Read(A)		

T1 bị huỷ và bắt đầu thực hiện với timestamp mới

Mặc dù trong quản lý giao tác 2 hành động đọc/ghi có tác dụng rất khác nhau

Không phân biệt tính chất của thao tác là đọc hay ghi → T1 vẫn bị hủy và làm lại từ đầu với 1 timestamp mớI

→ Sử dụng nhãn thời gian riêng phần

Nhãn thời gian riêng phần

- Nhãn của đơn vị dữ liệu X được tách ra thành 2 loại:
  - **RT(X)** – read time of X hay **R-TS(X)**
    - Ghi nhận TS(T) gần nhất (giao tác có nhãn thời gian lớn nhất) đọc X thành công
  - **WT(X)** – write time of X hay **W-TS(X)**
    - Ghi nhận TS(T) gần nhất (giao tác có nhãn thời gian lớn nhất) ghi X thành công

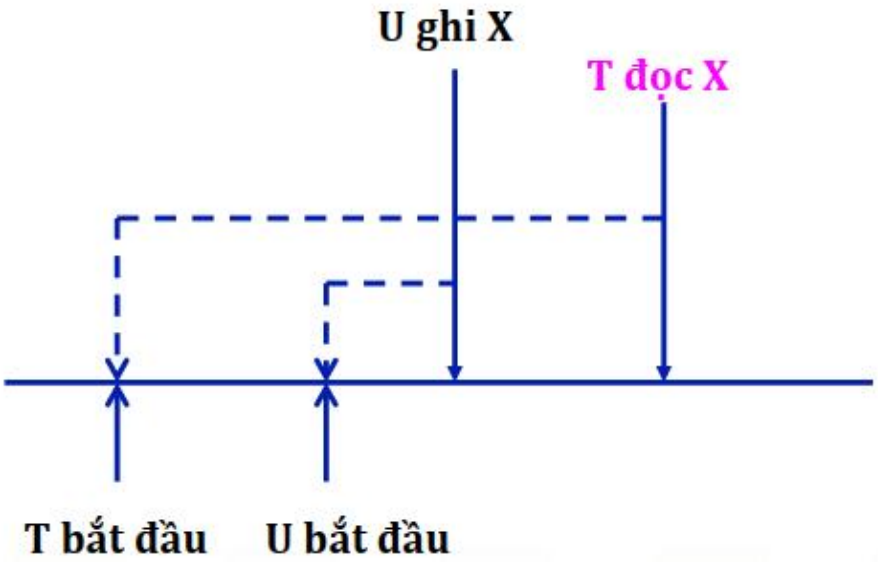
- Ngoài ra bộ lập lịch cũng quản lý trạng thái commit hay chưa của đơn vị dữ liệu X
  - C(X)= 1 nếu dữ liệu X đã được commit. Ngược lại C(X)=0
  - Bộ lập lịch dựa vào hoạt động của các giao tác để gán nhãn C(X) lên các đơn vị dữ liệu.
  - Kỹ thuật này được sử dụng để tránh việc lỗi đọc phải dữ liệu rác.

Nhãn thời gian riêng phần

- Công việc của bộ lập lịch:
  - Gán gắn các nhãn thời gian RT(X) và WT(X) và C(X)
  - Kiểm tra thao tác đọc/ghi xuất hiện khi nào để quyết định:
    - Chấp nhận yêu cầu
    - Trì hoãn giao tác
    - Huỹ bỏ giao tác
    - Bỏ qua hoạt động đó
  - Xử lý các tình huống:
    - Đọc quá trễ
    - Ghi quá trễ
    - Đọc dữ liệu rác
    - Quy tắc ghi Thomas

Nhãn thời gian riêng phần

- Đọc quá trễ

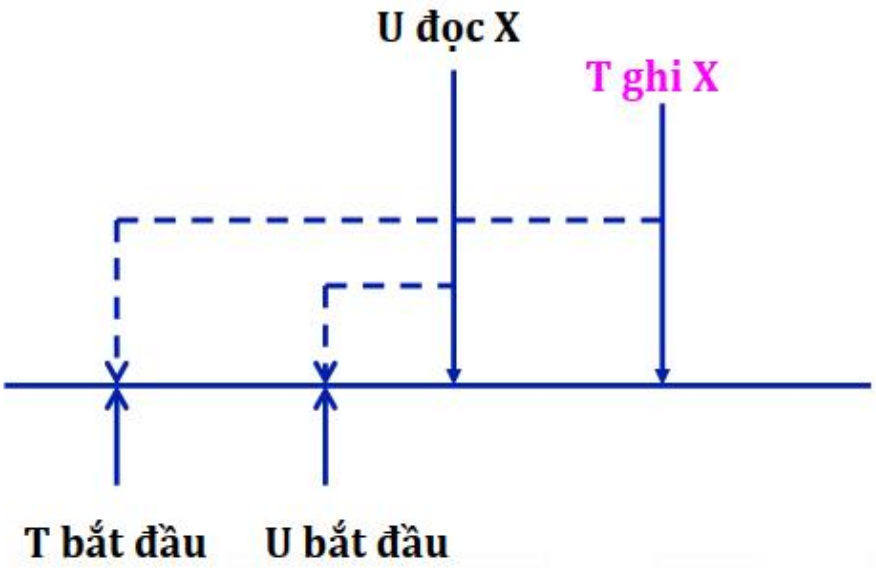


- T vào trước (TS(T) < TS(U)) muốn đọc X nhưng khi giá trị X hiện tại đã bị ghi bởi một giao tác khác vào sau T (TS(T) < WT(X))
- T không nên đọc X do U ghi bởi vì T vào trước

→ Giải pháp: Hủy T

Nhãn thời gian riêng phần

- Ghi quá trễ

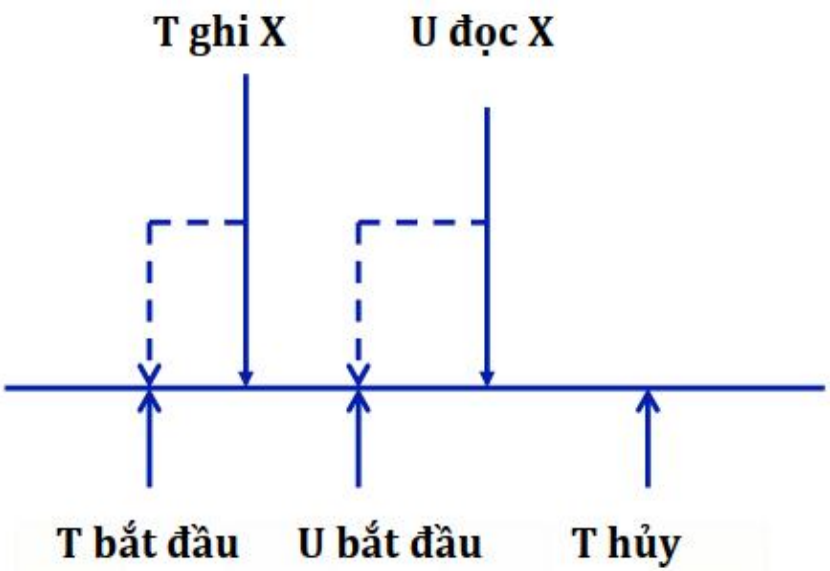


- T vào trước (TS(T) < TS(U)) và muốn ghi lên X, tuy nhiên X đã bị đọc bởi một giao tác vào sau T (WT(X) < TS(T) < RT(X))
- T không nên ghi X do giao tác U đã đọc X. (U phải đọc giá trị do T ghi)

→ Giải pháp: Hủy T

Nhãn thời gian riêng phần

- Đọc dữ liệu rác

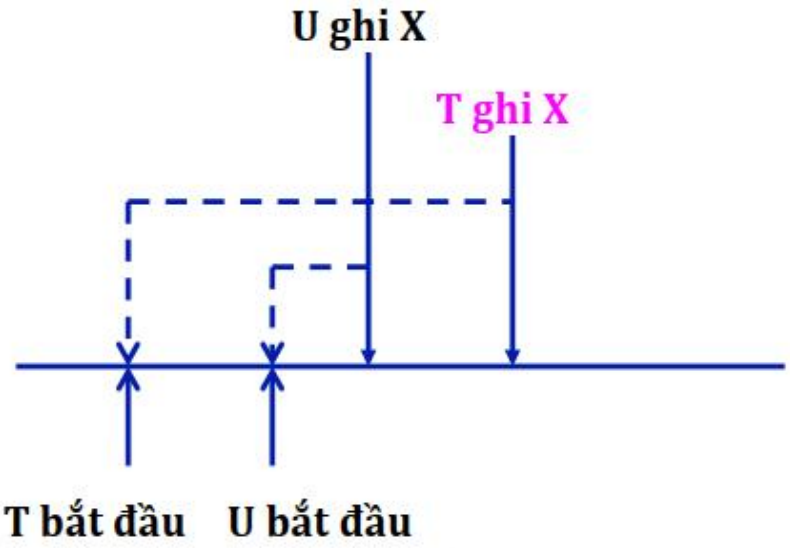


- T vào trước U và thực hiện việc ghi X trước. U vào sau và thực hiện việc đọc X.
- Nhưng T hủy → giá trị X mà U đọc là giá trị rác

→ Giải pháp: U đọc X sau khi T commit hoặc sau khi T abort.

Nhãn thời gian riêng phần

- Quy tắc ghi Thomas

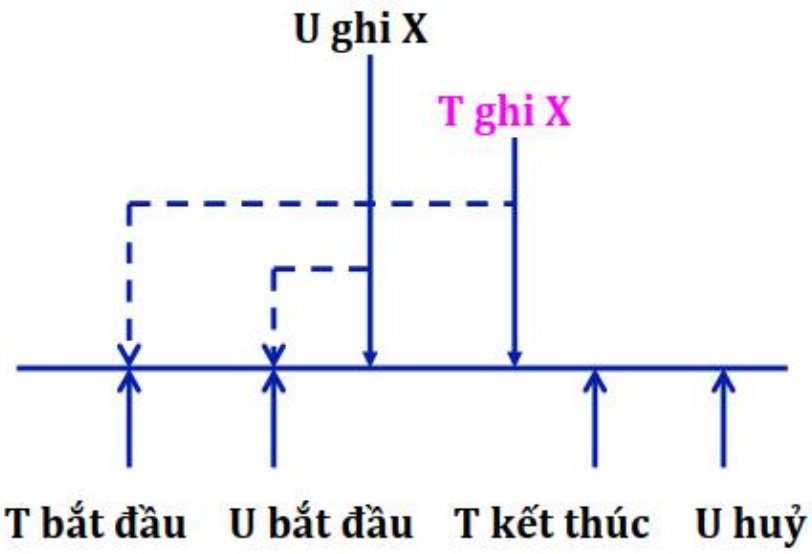


- T vào trước U vào sau nhưng khi T muốn ghi lên X thì U đã ghi lên X trước (TS(T) < WT(X))
- T nếu có ghi xong X cũng không làm được gì vì:
  - T không thể đọc X vì nếu đọc X thì sẽ dẫn đến đọc trễ
  - Các giao tác khác sau T và U thì muốn đọc giá trị X được ghi bởi U

→ Giải pháp: Bỏ qua thao thác ghi X của T [Quy tắc ghi Thomas]

Nhãn thời gian riêng phần

- Vấn đề với quy tắc ghi Thomas



- Trường hợp U ghi và sau đó bị huỷ → giá trị được ghi bởi U đã bị mất
- Do T đã kết thúc → cần khôi phục lại giá trị X từ T mà lệnh ghi X đã bị bỏ qua

→ Giải pháp: Khi T ghi X, nếu giao tác U đã commit thì bỏ qua T, hoặc đợi đến thời điểm U commit hoặc abort.

Nhãn thời gian riêng phần

- Tóm lại khi có yêu cầu đọc và ghi từ giao tác T. Bộ lập lịch sẽ:
  - Đáp ứng yêu cầu
  - Hủy T và khởi tạo lại T với 1 timestamp mới
    - T rollback
  - Trì hoãn T, sau đó mới quyết định phải hủy T hoặc đáp ứng yêu cầu

Nhãn thời gian riêng phần

- Quy tắc :
  - Nếu T cần đọc X
    - Nếu WT(X) ≤ TS(T) thì chờ cho X trở thành Dữ liệu đã Commit rồi cho T đọc X và gán RT(X) = MAX(RT(X), TS(T))
    - Ngược lại hủy T và khởi động lại T cới TS(T) mới (đọc quá trễ)
  - Nếu T cần ghi X
    - Nếu RT(X) ≤ TS(T)
      - Nếu WT(X) ≤ TS(T) thì cho T ghi X và gán WT(X) = TS(T)
      - Ngược lại thì bỏ qua thao tác ghi này của T (không hủy T)
    - Ngược lại hủy T và khởi động lại T với TS(T) mới (ghi quá trễ)

Nhãn thời gian riêng phần (tt)

- Quy tắc:

Read(T, X)

```
If WT(X) <= TS(T)
    Read(X);//cho phép đọc X
    RT(X):= max(RT(X),TS(T));
Else
    Rollback{T};   //hủy T và khởi tạo lại với TS mới
```

Write(T, X)

```
If RT(X) <= TS(T)
    If WT(X) <= TS(T)
        Write(X);//cho phép ghi X
        WT(X):= TS(T);
    Else
        Nếu X đã COMMIT → bỏ qua, ngược lại chờ cho
      đến khi giao tác thực hiện ghi trên X commit
      hoặc abort
Else
        Rollback{T};  //hủy T và khởi tạo lại với TS mới
```

Ví dụ

	T_1 $TS(T_1)=100$	T_2 $TS(T_2)=200$	A RT(A)=0 WT(A)=0	B RT(B)=0 WT(B)=0	C RT(C)=0 WT(C)=0	
1	Read(A)		RT(A)=100 WT(A)=0			$WT(A) < TS(T_1)$ T_1 đọc được A
2		Read(B)		RT(B)=200 WT(B)=0		$WT(B) < TS(T_2)$ T_2 đọc được B
3	Write(A)		RT(A)=100 WT(A)=100			$RT(A) < TS(T_1)$ $WT(A) = TS(T_1)$ — T_1 ghi lên A được
4		Write(B)		RT(B)=200 WT(B)=200		$RT(B) < TS(T_2)$ $WT(B) = TS(T_2)$ — T_2 ghi lên B được
5		Read(C)			RT(C)=200 WT(C)=0	$WT(B) < TS(T_2)$ T_2 đọc được C
6	Read(C)				RT(C)=200 WT(C)=0	$WT(C) < TS(T_1)$ T_1 đọc được C
7	Write(C)					$RT(C) > TS(T_1)$ T_1 không ghi lên C được

Abort

Ví dụ (tt)

T_1 TS=200	T_2 TS=150	T_3 TS=175	A RT=0 WT=0	B RT=0 WT=0	C RT=0 WT=0
Read(B)				RT=200 WT=0	
	Read(A)		RT=150 WT=0		
		Read(C)			RT=175 WT=0
Write(B)				RT=200 WT=200	
Write(A)			RT=150 WT=200		
	Write(C)				
		Write(A)			

Rollback

Giá trị của A đã sao lưu bởi T1 → T3 không bị rollback và không cần ghi A

Ví dụ (tt)

T_1 TS=150	T_2 TS=200	T_3 TS=175	T_4 TS=255	A RT=0 WT=0
Read(A)				RT=150 WT=0
Write(A)				RT=150 WT=150
	Read(A)			RT=200 WT=0
	Write(A)			RT=200 WT=200
		Read(A)		
			Read(A)	RT=255 WT=200

Rollback

T3 bị hủy vì nó định đọc giá trị A ghi bởi T2 (mà T2 lại có nhãn thời gian lớn hơn nó). Giả sử T3 đọc giá trị A ghi bởi T1 thì T3 sẽ không bị hủy

Ý tưởng lưu giữ nhiều phiên bản của A

Nhãn thời gian nhiều phiên bản

- Ý tưởng
  - Cho phép thao tác read(A) của T3 thực hiện
- Bên cạnh việc lưu trữ giá trị hiện hành của A, ta giữ lại các giá trị được sao lưu trước kia của A (phiên bản của A)
- Giao tác T sẽ đọc được giá trị của A ở 1 phiên bản thích hợp nào đó

Nhãn thời gian nhiều phiên bản (tt)

- Mỗi phiên bản của 1 đơn vị dữ liệu X có
  - RT(X)
    - Ghi nhận lại giao tác sau cùng đọc X thành công
  - WT(X)
    - Ghi nhận lại giao tác sau cùng ghi X thành công
- Khi giao tác T phát ra yêu cầu thao tác lên X
  - Tìm 1 phiên bản thích hợp của X
  - Đảm bảo tính khả tuần tự
- Một phiên bản mới của X sẽ được tạo khi hành động ghi X thành công

Nhãn thời gian nhiều phiên bản (tt)

- Quy tắc:

Read(T, X)

```
i="số thứ tự phiên bản sau cùng nhất của X"
While WT(Xi) > TS(T)
    i:=i-1;//lùi lại
Read(Xi);
RT(Xi):= max(RT(Xi), TS(T));
```

Write(T, X)

```
i="số thứ tự phiên bản sau cùng nhất của X"
While WT(Xi) > TS(T)
    i:=i-1;//lùi lại

If RT(Xi) > TS(T)
    Rollback T//Hủy và khởi tạo TS mới
Else
    Tạo phiên bản Xi+1;
    Write(Xi+1);
    RT(Xi+1) = 0;//chưa có ai đọc
    WT(Xi+1) = TS(T);
```

Ví dụ

T_1 TS=150	T_2 TS=200	T_3 TS=175	T_4 TS=255	A_0 RT=0 WT=0	A_1	A_2
Read(A)				RT=150 WT=0		
Write(A)					RT=0 WT=150	
	Read(A)				RT=200 WT=150	
	Write(A)					RT=0 WT=200
		Read(A)			RT=200 WT=150	
			Read(A)			RT=255 WT=200

Ví dụ (tt)

T_1 TS=100	T_2 TS=200	A_0 RT=0 WT=0	B_0 RT=0 WT=0	A_2	B_1
Read(A)		RT=100 WT=0			
	Write(A)		RT=0 WT=200	RT=0 WT=200	
	Write(B)				RT=0 WT=200
Read(B)				RT=100 WT=0	
Write(A)			RT=0 WT=100		

Nhãn thời gian nhiều phiên bản (tt)

- Nhận xét
  - Thao tác đọc
    - Giao tác T chỉ đọc giá trị của phiên bản do T hay những giao tác trước T cập nhật
    - T không đọc giá trị của các phiên bản do các giao tác sau T cập nhật

    → Thao tác đọc không bị rollback
  - Thao tác ghi
    - Thực hiện được thì chèn thêm phiên bản mới
    - Không thực hiện được thì rollback
  - Tốn nhiều chi phí tìm kiếm, tốn bộ nhớ
  - Nên giải phóng các phiên bản quá cũ không còn được các giao tác sử dụng

3. GIAO THỨC DỰA TRÊN TÍNH HỢP LỆ

GT dựa trên tính hợp lệ

- GT điều khiển cạnh tranh dựa trên tính hợp lệ còn được gọi là kỹ thuật điều khiển cạnh tranh lạc quan.
- Trong suốt quá trình các GD thực thi hệ thống không cần phải giám sát, chỉ thực hiện việc kiểm tra 1 số giai đoạn của GD.
- Một GD T_i được chia thành 2 hoặc 3 kỳ tùy thuộc vào GD là chỉ đọc hay vừa đọc vừa ghi.

GT dựa trên tính hợp lệ (tt)

- Các kỳ này, theo thứ tự như sau:
  - **Kỳ đọc:** Các hạng mục DL đọc vào các biến cục bộ tạm
  - **Kỳ hợp lệ:** Kiểm tra tính hợp lệ để xác định có thể cập nhật từ biến tạm lên CSDL hay không.
  - **Kỳ ghi:** Nếu kỳ hợp lệ thành công ⇒ cập nhật;

    Ngược lại ⇒ cuộn lại.

GT dựa trên tính hợp lệ (tt)

- Để kiểm tra kỳ hợp lệ, mỗi GD sẽ được gán các tem thời gian tương ứng với 3 kỳ của GD:
  - **Start(T_i):** Thời gian khi T_i bắt đầu thực hiện
  - **Validation(T_i):** Thời gian khi T_i kết thúc kỳ đọc và khởi động kỳ hợp lệ
  - **Finish(T_i):** Thời gian khi T_i kết thúc kỳ viết
- **$TS(T_i)$ = Validation(T_i)** và nếu $TS(T_i) < TS(T_j)$ //T_i trước T_j

Lịch trình hợp lệ ⇔ Lịch trình tuần tự

GT dựa trên tính hợp lệ (tt)

- **Phép kiểm thử tính hợp lệ đối với T_j** đòi hỏi mỗi GD T_i với $TS(T_i) < TS(T_j)$, một trong các điều kiện sau phải thỏa:
  - **$Finish(T_i) < Start(T_j)$:** Do T_i bắt đầu và kết thúc trước T_j, tính khả tuần tự được đảm bảo
  - **Tập các hạng mục DL được viết bởi T_i ($WS(T_i)$) không giao với tập các hạng mục DL được đọc bởi T_j ($RS(T_j)$) và T_i hoàn tất kỳ viết của nó trước khi T_j bắt đầu kỳ hợp lệ**

    $(Start(T_j) < Finish(T_i) < Validation(T_j))$

- Đảm bảo các kỳ ghi của T_i và T_j không chồng chéo nhau.

  ➔ Thứ tự khả tuần tự được duy trì

Ví dụ

Lịch trình S_3 dưới đây có hợp lệ hay không?

T_{14}	T_{15}
Read(B)	
	Read(B)
	B:=B-50
	Read(A)
	A:=A+50
Read(A)	
Xác nhận tính hợp lệ	
Display(A+B)	
	Xác nhận tính hợp lệ
	Write(B)
	Write(A)

Giả sử $TS(T_{14}) < TS(T_{15})$ thì:

- **Kỳ hợp lệ T_{14}:** do không tồn tại T_i nào $\Leftrightarrow TS(T_i)<TS(T_{14})$
- **Kỳ hợp lệ T_{15}:** ta có
  - $TS(T_{14}) < TS(T_{15})$
  - $WS(T_{14}) = \varnothing$, $RS(T_{15}) = \{A, B\}$

  $\Rightarrow WS(T_{14}) \cap RS(T_{15}) = \varnothing$
  - $Start(T_{15}) < Finish(T_{14}) < Validation(T_{15})$

LT S_3 - LT hợp lệ dưới giao thức dựa trên tính hợp lệ

4. QUẢN LÝ DEADLOCK

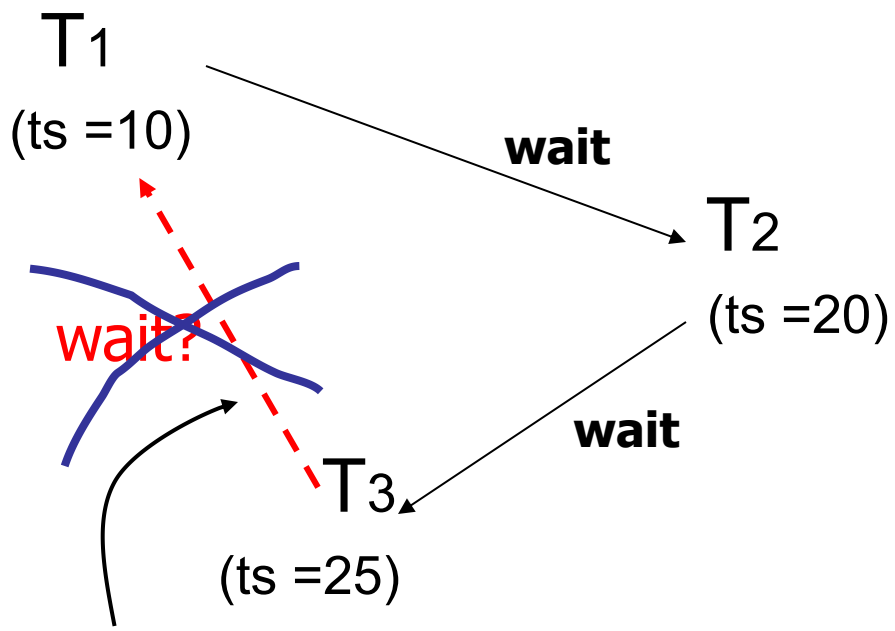
Deadlock

- **Deadlock:** hệ thống $\exists \{T_1, T_2, \ldots, T_n\}$ sao cho T_1 chờ T_2, T_2 chờ T_3, …T_{n-1} chờ T_n, T_n chờ T_1
  - ⇒ Không GD nào tiến triển được
  - ⇒ Phải "tẩy rửa" một vài GD tham gia vào deadlock
- Xử lý deadlock trong hệ thống:
  - Ngăn ngừa deadlock: phù hợp khi xác suất deadlock cao
    - Giao thức điều khiển CT ngăn ngừa deadlock (cây,…)
    - Wait-die
    - Wound-wait
    - **Time-out**
  - Phát hiện: phù hợp khi xác suất deadlock thấp
    - Wait-for graph

Ngăn ngừa deadlock

Wait-die:

- Một GD T_i được gán 1 tem thời gian $TS(T_i)$ khi bắt đầu
- T_i có thể chờ T_j nếu $TS(T_i) < TS(T_j)$
  Ngược lại, T_i "chết" và cuộn lại

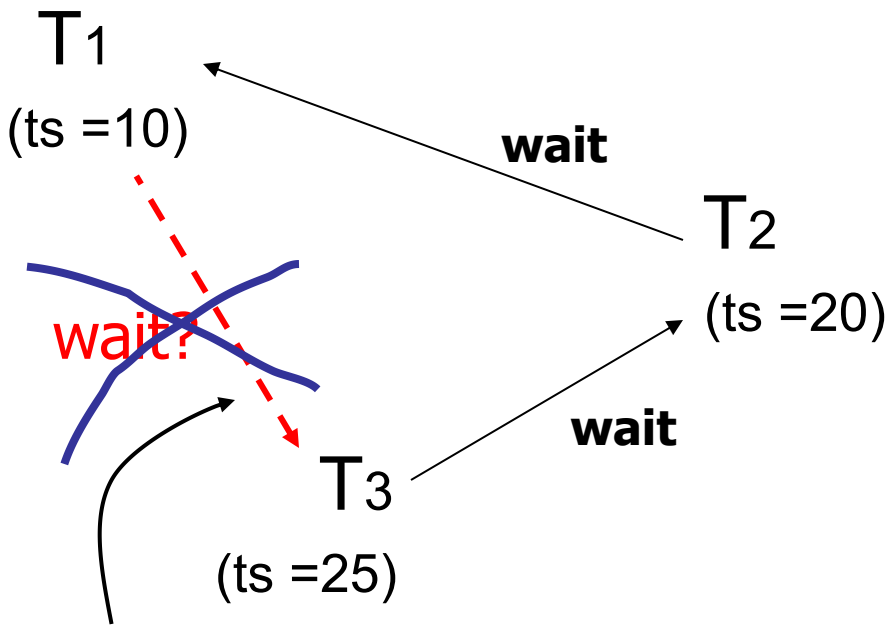


⇒ T3 không chờ được T1 ⇒ T3 chết và cuộn lại

Ngăn ngừa deadlock (tt)

Wound-wait:

- Một GD T_i được gán 1 tem thời gian $TS(T_i)$ khi bắt đầu
- T_i "ép chết" (wound) T_j nếu $TS(T_i) < TS(T_j)$
  Ngược lại, T_i chờ



⇒ T1 "ép chết" T3 ⇒ T3 chết và cuộn lại

Ngăn ngừa deadlock (tt)

- **Chú ý:**
  - Trong cả hai sơ đồ, khi một GD bị cuộn lại sẽ không được gán tem thời gian mới ⇒ tránh "chết đói".
  - Wait-die vs. Wound-wait:

Wait-die	Wound-wait
- GD già phải chờ GD trẻ. ⇒GD già có xu hướng chờ nhiều hơn.	- GD già không bao giờ chờ GD trẻ.
- Một GD T_i chết, sau khi khởi động lại có thể chết tiếp tục. ⇒T_i có thể chết vài lần trước khi tậu một hmục dl cần thiết.	- Một GD T_i bị thương và cuộn lại, sau khi khởi động lại sẽ chờ/ttục. ⇒ Có ít sự cuộn lại hơn trong sơ đồ Wound-wait.

Ngăn ngừa deadlock (tt)

Timeout:

- Sơ đồ trung gian giữa ngăn ngừa và phát hiện deadlock.
- Nếu một GD T_i chờ quá một khoảng thời gian **L**

  ⇒ T_i sẽ bị cuộn lại

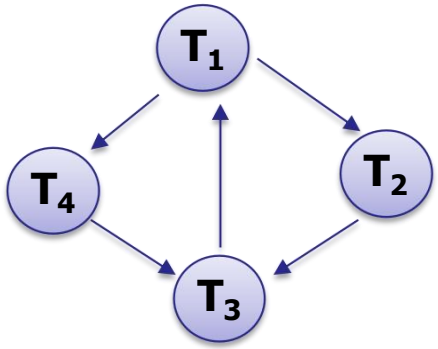
- Ưu điểm: đơn giản

- Khuyết điểm:
  - Khó xác định **L**
  - Có thể gây ra chết đói

Phát hiện deadlock

- **Đồ thị chờ** (wait-for graph): G = <V, E>
  - **V** = tập các đỉnh, gồm tất cả các giao dịch trong hệ thống
  - **E** = tập các cung. Nếu T_i chờ T_j

    ⇒ bổ sung một cung $T_i \longrightarrow T_j$ vào **E**



- **Phát hiện deadlock**: nếu đồ thị chờ tại một thời điểm có chu trình ⇒ hệ thống tại thời điểm đó bị deadlock

Phát hiện deadlock (tt)

- Bao lâu xây dựng đồ thị chờ một lần?
  - Deadlock thường xảy ra không?
  - Bao nhiêu giao dịch sẽ bị ảnh hưởng bởi deadlock?
- Khôi phục từ deadlock:
  - Chọn nạn nhân: dựa trên các tiêu chí
    - thời gian thực hiện GD
    - số hạng mục dữ liệu sử dụng bởi GD
    - số giao dịch sẽ bị ảnh hưởng
  - Cuộn lại: cuộn lại bao xa
  - Sự chết đói: khi chọn nạn nhân, phải xem số lần đã cuộn lại GD

HẾT CHƯƠNG 4