

Trabalho de implementação: 2PL Strict

Lucas Martins de Barros

07/11/2024

Execução

O programa foi desenvolvido em *Python*. É utilizada uma biblioteca chamada “[tabulate](#)” para realizar prints das tabelas formatadas. A biblioteca pode ser instalada através do comando:

```
pip3 install tabulate
```

Com isso, é possível executar o programa através do arquivo “*main*”.

Entrada de dados

Para facilitar o uso existem cinco histórias pré-definidas, porém, é possível digitar uma história inicial manualmente. As histórias seguem um formato um pouco diferente do visto em aula, foi adicionado um atributo de “valor” quando a operação for write. Elas seguem o padrão:

```
s1 r1[x] s2 r1[y] w1[x,20] r2[y] c1 w2[x,10] c2
```

Onde o valor após a vírgula dentro do colchetes do *write* é o valor que será escrito na variável.

A visão inicial do programa é apresentada na figura a seguir:

Selecione uma transação de exemplo

```
0. Personalizar
1. Sem conflitos: s1 r1[x] s2 r1[y] w1[x,20] r2[y] c1 w2[x,10] c2
2. Com delay:    s1 r1[x] w1[x,10] s2 w2[x,15] c1 c2
3. Com deadlock: s1 s2 r1[x] w2[y,10] r1[y] w2[x,20] c1 c2
4. Erro de commit: s1 s2 r1[x] r2[y] r1[y] c1 r1[x] w2[x,10] c2
5. Erro de start: r1[x]
□
```

Os exemplos são:

1. História sem nenhum conflito, que roda do início ao fim sem *delays* e com um upgrade.
2. História com um *delay* e uma situação de upgrade, mas sem *deadlocks*.
3. História com dois *delays* em transações diferentes e, consequentemente, um *deadlock*.
4. História com erro, onde uma transação sofre *commit* (consequentemente *unlocks*) e depois realiza outra operação que necessita de um *lock*. É retornado um “Erro de *commit*”.
5. História com erro em que é realizada uma operação sem iniciar a transação, retornando um “Erro de *start*”.

Estrutura de dados

Para armazenar os dados do escalonador são usadas 3 tabelas: locks, operações e transações. Uma explicação e uma figura de exemplo das tabelas podem ser vistos a seguir. Além das tabelas, foi necessário uma lista de “Operações final” para gerar a história

final. As tabelas são listas de dicionários do Python, transformadas em tabelas para facilitar na visualização (usando a biblioteca *tabulate*).

Tabela de locks

A tabela de locks possui uma variável, valor (padrão é sempre 0), lista de transações com bloqueio compartilhado (padrão é em branco) e transação com bloqueio exclusivo (padrão é *None*).

var	valor	ls	lx
y	0	[]	
x	0	[]	

Tabela de operações

As operações se tornaram uma classe, com os atributos iguais às colunas da tabela:

- Tipo de operação: *start* (s), *commit* (c), *read* (r) ou *write* (w). Para “operações finais” também é possível existir os tipos *lock* (l) ou *unlock* (u).
- id_transacao.
- variável: se for uma operação de *write*, *read*, *unlock* ou *lock*.
- valor: se for uma operação de *write* irá conter o valor escrito, e para operações de *lock* e *unlock* contém o tipo de *lock* (s ou x).
- Status: ok, pendente ou *delayed*.
- Tentativas: inicia com 0 e, a cada tentativa sem sucesso de executar (*delayed*) soma 1.

tipo_operacao	id_transacao	variavel	valor	status	tentativas
s	1			Ok	0
s	2			Pendente	0
r	1	x		Pendente	0
w	2	y	10	Pendente	0
r	1	y		Pendente	0
w	2	x	20	Pendente	0
c	1			Pendente	0
c	2			Pendente	0

Tabela de transações

A tabela é composta por transação e status, sendo que o status será “Não iniciada”, “Iniciada” ou “Delayed”.

Transação	Status
2	Não iniciada
1	Iniciada

Características de funcionamento

Na terceira tentativa de executar uma operação que sofreu delay é determinado um *deadlock*. O tratamento do *deadlock* é realizado movendo uma das transações envolvidas no *deadlock* para o final da lista de operações e reiniciando a transação, removendo registros da história final e mudando os status de das operações que tinham sido realizadas para "pendente". Dessa forma, é removido o entrelaçamento de uma das transações envolvidas no *deadlock*.

Exemplo de execução

A seguir, um passo a passo da execução do exemplo 3 (com *deadlock*).

1. Inicia a transação 1 e transação 2:

Tabela de transações

Transação	Status
2	Iniciada
1	Iniciada

Tabela de locks

var	valor	ls	lx
y	0	[]	
x	0	[]	

Tabela de operações

tipo_operacao	id_transacao	variavel	valor	status	tentativas
s	1			Ok	0
s	2			Ok	0
r	1	x		Pendente	0
w	2	y	10	Pendente	0
r	1	y		Pendente	0
w	2	x	20	Pendente	0
c	1			Pendente	0
c	2			Pendente	0

HI: s1 s2 r1[x] w2[y,10] r1[y] w2[x,20] c1 c2
HF: s1 s2



2. Transação 1 realiza bloqueio compartilhado do X e leitura do mesmo:

Tabela de transações

Transação	Status
2	Iniciada
1	Iniciada

Tabela de locks

var	valor	ls	lx
y	0	[]	
x	0	['1']	

Tabela de operações

tipo_operacao	id_transacao	variavel	valor	status	tentativas
s	1			Ok	0
s	2			Ok	0
r	1	x		Ok	0
w	2	y	10	Pendente	0
r	1	y		Pendente	0
w	2	x	20	Pendente	0
c	1			Pendente	0
c	2			Pendente	0

HI: s1 s2 r1[x] w2[y,10] r1[y] w2[x,20] c1 c2
 HF: s1 s2 ls1[x] r1[x]



3. Transação 2 realiza bloqueio exclusivo do y e escreve o valor “10” no mesmo:

Tabela de transações

Transação	Status
2	Iniciada
1	Iniciada

Tabela de locks

var	valor	ls	lx
y	10	[]	2
x	0	['1']	

Tabela de operações

tipo_operacao	id_transacao	variavel	valor	status	tentativas
s	1			Ok	0
s	2			Ok	0
r	1	x		Ok	0
w	2	y	10	Ok	0
r	1	y		Pendente	0
w	2	x	20	Pendente	0
c	1			Pendente	0
c	2			Pendente	0

HI: s1 s2 r1[x] w2[y,10] r1[y] w2[x,20] c1 c2
 HF: s1 s2 ls1[x] r1[x] lx2[y] w2[y,10]



4. Transação 1 tenta realizar a leitura do y, mas não consegue por estar com bloqueio exclusivo pela transação 2. Operação é colocada em “*Delayed*” e o número de tentativas aumenta para 1:

Tabela de transações

Transação	Status
2	Iniciada
1	Delayed

Tabela de locks

var	valor	ls	lx
y	10	[]	2
x	0	['1']	

Tabela de operações

tipo_operacao	id_transacao	variavel	valor	status	tentativas
s	1			Ok	0
s	2			Ok	0
r	1	x		Ok	0
w	2	y	10	Ok	0
r	1	y		Delayed	1
w	2	x	20	Pendente	0
c	1			Pendente	0
c	2			Pendente	0

HI: s1 s2 r1[x] w2[y,10] r1[y] w2[x,20] c1 c2
 HF: s1 s2 ls1[x] r1[x] lx2[y] w2[y,10]

5. Transação 2 tenta realizar a leitura do x, porém a transação 1 está com bloqueio compartilhado do mesmo. A operação também é colocada em “*Delayed*” e o número de tentativas aumenta para 1:

Tabela de transações

Transação	Status
2	Delayed
1	Delayed

Tabela de locks

var	valor	ls	lx
y	10	[]	2
x	0	['1']	

Tabela de operações

tipo_operacao	id_transacao	variavel	valor	status	tentativas
s	1			Ok	0
s	2			Ok	0
r	1	x		Ok	0
w	2	y	10	Ok	0
r	1	y		Delayed	1
w	2	x	20	Delayed	1
c	1			Pendente	0
c	2			Pendente	0

HI: s1 s2 r1[x] w2[y,10] r1[y] w2[x,20] c1 c2
 HF: s1 s2 ls1[x] r1[x] lx2[y] w2[y,10]

6. São realizadas mais 2 tentativas de executar as operações. Na terceira tentativa é notificado o *deadlock*:

Tabela de transações

Transação	Status
1	Delayed
2	Delayed

Tabela de locks

var	valor	ls	lx
x	0	['1']	
y	10	[]	2

Tabela de operações

tipo_operacao	id_transacao	variavel	valor	status	tentativas
s	1			Ok	0
s	2			Ok	0
r	1	x		Ok	0
w	2	y	10	Ok	0
r	1	y		Delayed	3
w	2	x	20	Delayed	3
c	1			Pendente	0
c	2			Pendente	0

HI: s1 s2 r1[x] w2[y,10] r1[y] w2[x,20] c1 c2

HF: s1 s2 ls1[x] r1[x] lx2[y] w2[y,10]

DEADLOCK: r1[y] w2[x,20]

□

7. Para tratar o deadlock, a transação 2 é reiniciada, removendo todas operações já realizadas por ela da história final, removendo seus *locks* da tabela de *locks*, voltado os valores modificados por ela para o valor antes da modificação (nesse caso o y vira 0 novamente) e movendo todas suas operações para o fim da ordem de execução:

Tabela de transações

Transação	Status
1	Delayed
2	Não iniciada

Tabela de locks

var	valor	ls	lx
x	0	['1']	
y	0	[]	

Tabela de operações

tipo_operacao	id_transacao	variavel	valor	status	tentativas
s	1			Ok	0
r	1	x		Ok	0
r	1	y		Delayed	3
c	1			Pendente	0
s	2			Pendente	0
w	2	y	10	Pendente	0
w	2	x	20	Pendente	0
c	2			Pendente	0

HI: s1 s2 r1[x] w2[y,10] r1[y] w2[x,20] c1 c2

HF: s1 ls1[x] r1[x]

■

8. Com isso, as operações começam do início novamente, realizando a operação da transação 1 que estava “*Delayed*”:

Tabela de transações

Transação	Status
1	Iniciada
2	Não iniciada

Tabela de locks

var	valor	ls	lx
x	0	['1']	
y	0	['1']	

Tabela de operações

tipo_operacao	id_transacao	variavel	valor	status	tentativas
s	1			Ok	0
r	1	x		Ok	0
r	1	y		Ok	3
c	1			Pendente	0
s	2			Pendente	0
w	2	y	10	Pendente	0
w	2	x	20	Pendente	0
c	2			Pendente	0

HI: s1 s2 r1[x] w2[y,10] r1[y] w2[x,20] c1 c2
 HF: s1 ls1[x] r1[x] ls1[y] r1[y]

■

9. Após a última operação de leitura da transação 1 ela sofre o “*commit*”, que possibilita desbloquear as duas variáveis que estava bloqueando:

Tabela de transações

Transação	Status
1	Finalizada
2	Não iniciada

Tabela de locks

var	valor	ls	lx
x	0	[]	
y	0	[]	

Tabela de operações

tipo_operacao	id_transacao	variavel	valor	status	tentativas
s	1			Ok	0
r	1	x		Ok	0
r	1	y		Ok	3
c	1			Ok	0
s	2			Pendente	0
w	2	y	10	Pendente	0
w	2	x	20	Pendente	0
c	2			Pendente	0

HI: s1 s2 r1[x] w2[y,10] r1[y] w2[x,20] c1 c2
 HF: s1 ls1[x] r1[x] ls1[y] r1[y] c1 us1[x] us1[y]

■

10. Depois do *commit* da transação 1, a transação 2 é iniciada, executando sem erros até seu *commit*:

Tabela de transações

Transação	Status
2	Finalizada
1	Finalizada

Tabela de locks

var	valor	ls	lx
y	10	[]	
x	20	[]	

Tabela de operações

tipo_operacao	id_transacao	variavel	valor	status	tentativas
s	1			Ok	0
r	1	x		Ok	0
r	1	y		Ok	3
c	1			Ok	0
s	2			Ok	0
w	2	y	10	Ok	0
w	2	x	20	Ok	0
c	2			Ok	0

HI: s1 s2 r1[x] w2[y,10] r1[y] w2[x,20] c1 c2

HF: s1 ls1[x] r1[x] ls1[y] r1[y] c1 us1[y] us1[x] s2 lx2[y] w2[y,10] lx2[x] w2[x,20] c2 ux2[y] ux2[x]

Código

O código pode ser acesso pelo repositório no [Github](#)