

Trabalho 11

Compare detalhadamente os modelos de consistência para sistemas paralelos discutidos ao longo da disciplina. Devem ser usados exemplos e discutidas questões como semelhanças e diferenças entre os modelos, as garantias por eles fornecidas, aplicações possíveis e o quão mais fácil é para um programador usar cada um (supondo que a linguagem ou infra-estrutura subjacentes garantam isso). Os modelos foram os seguinte:

=====

Tabela Comparativa de Modelos de Memória

	Quiescent	Sequential	Linearizability	Serializability	Happens before
Equivale a ordem sequencial	+	+	+	+	+
Respeita a ordem do programa	-	+	+	+	+
Consistente com execução tempo real	+ (quando há período de quiescência)	-	+	-	+
Operação pode afetar múltiplos objetos	-	-	-	+	+
Compositinal	+	-	+	-	+
Não bloqueante	+	+	+	-	-

=====

● Consistência Quiescente:

1. Definições:

Quiescent consistency is similar to linearizability, except that the response-to-invocation order must be respected only across a quiescent point, that is, a point with no open method calls. -> FONTE: **Between Linearizability and Quiescent Consistency**

Quiescent consistency is an alternative criterion which guarantees that a concurrent data structure behaves correctly when accessed sequentially. Yet quiescent consistency says very little about executions that have any contention. -> FONTE: **Between Linearizability and Quiescent Consistency**

Any time an object becomes quiescent, execution so far is equivalent to some sequential execution of the completed calls. FONTE: **AMP**

It is non-blocking and compositional

2. Exemplos:

execução sequencial: $[]^1 ()^2 \{ \}^3 < >^4$

exemplo quiescente 1: $[()^2 \{ < \}^3 >^4]^1$

exemplo quiescente 2: $[([]^1 \{ \}^2 < \}^3 >^4$

exemplo quiescente 3: $[([]^1)^2 \{ < \}^3 >^4$

exemplo não quiescente: $[(< []^1)^2 >^4 \{ \}^3$

Obs: cada cor acima é uma thread, e o lado esquerdo do "parênteses" é o início da execução e o lado direito o fim. E cada número indica a ordem de chamada de cada thread.

>> consistência de quietude mas não a sequencial:

```
A-----|write(x) write(z)|----|read(x)|-----
B-----|write(y)|-----
```

3. Semelhanças e Diferenças:

Não há semelhança entre sequencial e quiescente. Visto que Quiescent não necessariamente preserva a ordem do programa assim como Sequential não é afetado pelo período de quiescência.

Se assemelha com o Linearizability, exceto que a ordem das chamadas deve ser respeitada somente até depois do ponto de quiescência.

Se assemelha com serializable, pois não há garantia de ordem para transações com overlapping desde que o resultado final seja equivalente a execução serial das transações (em sequência).

O happens-before não pode ter a ordem de execução alterada devido o relacionamento que determina a ordem de execução, diferente de Quiescent onde não há garantia da ordem de execução quando não há período de quiescência.

4. Garantias fornecidas:

Compositionality: execuções com consistência de quiescência podem ser combinadas e o resultado continua com consistência de quiescência.

É não-bloqueante.

Todas as operações aparecem ocorrer de forma sequencial; operações que não ocorrem overlapping aparecem ocorrer na ordem de tempo real. Quando há um período de quiescência, as operações não separadas por ele (as que aconteceram antes deste período) podem não acontecer na ordem do programa.

5. Aplicações possíveis:

Apropriado para aplicações que requerem alto desempenho com um custo de colocar restrições relativamente baixas sobre o comportamento do objeto.

6. Facilidade de uso:

Como esta consistência é garantida pela máquina virtual de Java, esta operação torna-se transparente para o usuário

● Consistência Sequencial:

1. Definições:

Definition:[A multiprocessor system is sequentially consistent if] the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.

There are two aspects to sequential consistency: (1) maintaining program order among operations from individual processors, and (2) maintaining a single sequential order among operations from all processors.

Sequential consistency is a consistency condition weaker than linearizability. For counting networks, it assures that the order of values returned to the same process reflects the realtime

order in which the values were requested. This natural monotonicity property is reasonable to expect from a counter primitive. **FONTE - Sequentially Consistent versus Linearizable Counting Networks***

Informally, sequential consistency requires that all memory operations appear to execute one at a time, and the operations of a single processor appear to execute in the order described by that processor's program

2. Exemplos:

>> sequencialmente consistente mas não tem consistência de quietude:

```
A----|write(x)=a|-----
B-----|write(x)=b|-----
c-----|read(x)=b|-----|read(x)=a|-----
d-----|read(x)=b|----|read(x)=a|-----
```

>> Zookeeper (Apache) - Um Servidor de Coordenação para Aplicações Distribuídas: as atualizações dos clientes são aplicadas na ordem em que elas foram enviadas.

3. Semelhanças e Diferenças:

Não há semelhança entre sequencial e quiescente. Visto que Quiescent não necessariamente preserva a ordem do programa assim como Sequential não é afetado pelo período de quiescência.

Toda execução linearizable é sequencialmente consistente, mas o contrário não é verdadeiro.

Em operações Serializability, quando há períodos de quiescência, é possível garantir o resultado final (a ordem do programa é respeitada), enquanto que sequencial não é possível ter essa garantia da ordem entre as threads.

A consistência sequencial só se assemelha com Happens-before nas operações realizadas na mesma thread, isto é, as operações seguem uma ordem pré-estabelecidas.

Este tipo de consistencia reduz a possibilidade de otimizações de desempenho comumente utilizadas por compiladores e processadores, reduzindo assim os benefícios dos multiprocessadores. Para contornar este problema, muitos multiprocessadores suportam modelos de memória mais flexíveis.

4. Garantias fornecidas:

Compositionality: execuções com consistência sequencial combinadas não mantêm a consistência de sequencial.

É não-bloqueante.

Consistência sequencial fornece um modelo de programação simples e intuitivo, Entretanto, esta consistência não permite otimizações tanto de hardware como do compilador (porque não se pode reordenar as operações dentro da thread).

Todas as operações aparecem ocorrer de forma sequencial; a ordem é consistente com a ordem de programa de cada thread. Não é afetada por tempos de quiescência.

5. Aplicações possíveis:

Aplicações onde a composicionalidade dos componentes não seja um requisito.

6. Facilidade de uso:

As operações de leitura e escrita na memória não são sequencialmente consistente (o compilador pode reordenar as operações), desta forma o programador é obrigado a implementar esta consistência de maneira explícita utilizando instruções especiais de barreira de memória.

- **Linearizabilidade:**

1. Definições:

Linearizable => this is what usually understood by "thread-safe" -> FONTE:
<http://coldattic.info/shvedsky/pro/blogs/a-foo-walks-into-a-bar/posts/88>

A standard consistency condition, linearizability, requires that the order of the values returned to (possibly different) processes reflect the real-time order of the request operations [HW90]. That is, a request operation that was completed earlier must obtain a smaller value than one invoked after its completion. Linearizable counting can be used as a building block in basic constructions such as barrier synchronization* [MS91], concurrent queues and stacks [FG91] and efficient shared program counters [ML89]. -> FONTE ->

Sequentially Consistent versus Linearizable Counting Networks

Linearizability states roughly that every response-to-invocation order in a concurrent execution must be consistent with the sequential specification. -> FONTE: **Between Linearizability and Quiescent Consistency**

Desvantagens:

Unfortunately, linearizability imposes a performance penalty which scales linearly in the number of contending threads. Linearizability has proven quite useful in reasoning about concurrent executions; however, it fundamentally constrains efficiency in a multicore setting: Dwork, Herlihy, and Waarts [6] show that if many threads concurrently access a linearizable counter, there must be either a location with high

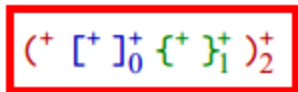
contention or an execution path that accesses many shared variables. -> FONTE: **Between Linearizability and Quiescent Consistency**

Linearizability is a compositional condition: a system of objects is linearizable if and only if each individual object is linearizable [HW90]. Sequential consistency is not a compositional condition. FONTE -> **Sequentially Consistent versus Linearizable Counting Networks**

It is non-blocking and compositional

2. Exemplos:

exemplo linear 1: $[([]^2 \{ < \}^3 >^4$



3. Semelhanças e Diferenças:

Toda execução linearizable é sequencialmente consistente, mas o contrário não é verdadeiro.

Se assemelha com o Quiescente, pois pode haver reordenação de métodos que tem overlapping para que exista uma execução consistente. O Quiescente ainda dita que a ordem das chamadas deve ser respeitada somente até depois do ponto de quiescência.

Enquanto que Serializability é uma propriedade global (transações), Linearizability é uma propriedade local (única transação). Outra diferença entre Linearizability e Serializability é que no primeiro o resultado final deve manter a ordem da execução, enquanto que no segundo o resultado pode variar. Outra diferença entre o Linearizability e o Serializability é que no primeiro o resultado ao método deve ter efeito de maneira imediata, enquanto que no segundo só tem efeito no final e toda a transação.

Tanto para o linear quanto para o happens-before, não é possível realizar reordenações das execuções dentro da mesma thread. Já no caso do linear, pode haver reordenação entre as threads.

4. Garantias fornecidas:

Linearizability é compositional. Duas execuções lineares combinadas, formam uma execução linear. Ou seja, esta propriedade garante que componentes lineares sejam combinados para formar sistemas fortemente consistentes.

Não bloqueante.

As operações entre as threads podem sofrer reordenação do modo que a operação final apresente um resultado possível. Pois cada chamada de método parece ter o efeito instantâneo em algum ponto entre a chamada e o retorno do método, desta forma é possível "redefinir" a ordem em que os métodos que tem overlapping tenham o seu efeito apresentado.

5. Aplicações possíveis:

Em aplicações standalone, como hardware de memória, onde composicionalidade não é um problema. Assim como em grandes sistemas, onde cada componente deve ser verificado e implementado independentemente.

6. Facilidade de uso:

Linearizability é uma poderosa ferramenta de especificação para objetos concorrentes. Além disso, permite a noção dos objetos serem “atômicos”.

- **Serializabilidade:**

1. Definições:

A transaction is a sequence of steps executed by a single thread. Transactions must be serializable, meaning that they appear to execute sequentially, in a one-at-a-time order. Serializability is a kind of coarse-grained version of linearizability. Linearizability defined atomicity of individual objects by requiring that each method call of a given object appear to take effect instantaneously between its invocation and response,

Serializability, on the other hand, **defines atomicity for entire transactions, that is, blocks of code that may include calls to multiple objects**. It ensures that a transaction appears to take effect between the invocation of its first call and the response to its last call.² Properly implemented, transactions do not deadlock or livelock. -> **Fonte -> LIVRO art of...**

Serializabilidade é a propriedade de o efeito da execução de diversas ações em paralelo ser equivalente ao efeito da execução sequencial destas ações em pelo menos uma de suas ordenações.

A importância deste conceito é que um conjunto de ações cujo efeito é serializável é atômico, o que significa que *todos* os seus possíveis comportamentos podem ser inferidos a partir da codificação original do programa, isto é, o programa paralelo tem a sua semântica definida.

execution is **serializable** if *its methods in each thread are grouped in contiguous, non-overlapping transactions, and the methods can be correctly ordered in such a way that transactions are still contiguous*.

- Serializability provides the same guarantees as linearizability, however, it does not impose any restrictions based the execution order of transactions. That is, the final state of the execution can be equivalent to any serial execution of transactions, no matter which transaction completes earlier.

Serializability x Linearizability

The central distinction between the two is that serializability is a *global* property; a property of an entire history of operations/transactions. Linearizability is a *local property*; a property of a single

operation/transaction. Another distinction is that linearizability includes a notion of real-time, which serializability does not: the linearization point of an operation must lie between its invocation and response times. (See Tim Harris: *Transactional Memory*, 2ed. See Herlihy's slides from *The Art of Multiprocessor Programming*, the section on *Linearizability*, which are [available here](#), for some examples and proofs. **Fonte** -> <http://stackoverflow.com/questions/4179587/difference-between-linearizability-and-serializability>

Serializability of a schedule means equivalence (in the outcome, the database state, data values) to a *serial schedule* (i.e., sequential with no transaction overlap in time)

<http://www.unine.ch/transact08/papers/Aydonat-Serializability.pdf>

http://pages.cs.wisc.edu/~dbbook/openAccess/thirdEdition/slides/slides3ed-english/Ch17_CC.pdf

http://www.eenadupratibha.net/pratibha/engineering/content_three_cse_process_synchronization.html
<http://www.cburch.com/cs/340/reading/serial/index.html>

2. Exemplos:

T1 -> Realiza subtração de 100 da conta A.

T2 -> Acrescenta 5% do total das conta A e B.

Considere: deve-se subtrair 100 da conta A e depois acrescentar 5% nas duas contas.

Schedule: $r_2(B)$, $w_2(B)$, $r_1(A)$, $w_1(A)$, $r_2(A)$, $w_2(A)$,

	A	B
(inicial):	200	100
$r_2(B)$:		
$w_2(B)$:		105
$r_1(A)$:		
$w_1(A)$:	100	
$r_2(A)$:		
$w_2(A)$:	105	

3. Semelhanças e Diferenças:

Se assemelha com quiescente, pois não há garantia de ordem para transações com overlapping desde que o resultado final seja equivalente a execução serial das transações (em sequência).

Em operações Serializability, quando há períodos de quiescência, é possível garantir o resultado final (a ordem do programa é respeitada), enquanto que Sequencial não é possível ter essa garantia da ordem entre as threads.

Enquanto que Serializability é uma propriedade global (transações), Linearizability é uma propriedade local (única transação). Outra diferença entre Linearizability e Serializability é que no primeiro o resultado final

deve manter a ordem da execução, enquanto que no segundo o resultado pode variar. Outra diferença entre o Linearizability e o Serializability é que no primeiro o resultado ao método deve ter efeito de maneira imediata, enquanto que no segundo só tem efeito no final e toda a transação.

Enquanto que Serializability pode ter a ordem da execução das operações alterada, desde que o resultado final seja equivalente a uma execução serial, em happens-before o reordenamento das operações não é permitido.

4. Garantias fornecidas:

Garante que resultado final das operações seja consistente (a corretude da execução de transações concorrentes) e que as transações sejam “atômicas”. Caso não seja serializável, uma saída errada pode ocorrer.

Garante dead-lock freedom e live-lock freedom.

5. Aplicações possíveis:

Sistemas críticos, como por exemplo sistemas bancário. E sistemas baseados em memória transacional, como por exemplo banco de dados.

6. Facilidade de uso:

- **Consistência Happens-before:**

1. Definições:

São operações que são previamente definidas para serem executadas em uma ordem. Definida a ordem de execução, elas não podem ser executadas fora desta ordem.

The results of a write by one thread are guaranteed to be visible to a read by another thread only if the write operation *happens-before* the read operation. The synchronized and volatile constructs, as well as the Thread.start() and Thread.join() methods, can form *happens-before* relationships

2. Exemplos:

unlock -> lock;

Thread.start -> qualquer operação da thread;

instanciação de qualquer objeto (não estático) -> chamada de método de qualquer objeto (não estático);

3. Semelhanças e Diferenças:

O happens-before não pode ter a ordem de execução alterada devido o relacionamento que determina a ordem de execução, diferente de Quiencent onde não há garantia da ordem de execução quando não há período de quiescência.

A consistência sequencial só se assemelha com Happens-before nas operações realizadas na mesma thread, isto é, as operações seguem uma ordem pré-estabelecidas.

Tanto para o linear quanto para o happens-before, não é possível realizar reordenações das execuções dentro da mesma thread. Já no caso do linerar, pode haver reordenação entre as threads.

Enquanto que Serializability pode ter a ordem da execução das operações alterada, desde que o resultado final seja equivalente a uma execução serial, em happens-before o reordenamento das operações não é permitido.

Caso Java fosse utilizar consistência sequencial como modelo de memória, muitas das otimizações que o compilador e o processador fazem seriam ilegais.

4. Garantias fornecidas:

É uma forte garantia para os desenvolvedores, pois eles não precisam se preocupar com condições de corrida (se o código estiver bem sincronizado). Somente nos casos em que o código não esteja bem sincronizado que eles precisam fazer o “tratamento” desta sincronização com as primitivas de Java.

5. Aplicações possíveis:

Qualquer aplicação que necessite de uma rigidez quanto a ordem de execução das suas operações.

6. Facilidade de uso:

Java provê mecanismos nativos que já implementam um relacionamento happens-before, como por exemplo: Thread.start(), Thread.join(). Java também fornece outras construções para que o programador crie relacionamentos happens-before, como por exemplo através de barreiras de memória.

Links úteis:

<http://coldattic.info/shvedsky/pro/blogs/a-foo-walks-into-a-bar/posts/88>

https://www.cs.rochester.edu/~scott/458/notes/04-concurrent_data_structures

<http://fmt.cs.utwente.nl/courses/cdp/slides/cdp-4-mem-4up.pdf>

<http://docs.oracle.com/javase/tutorial/essential/concurrency/memconsist.html>

<http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/package-summary.html#MemoryVisibility>