COMPARING CONSISTENCY MODELS

A memory consistency model dictates how memory behaves with respect to read and write operations from multiple concurrent processes. The consistency model of a shared-memory multiprocessor provides a formal specification of how the memory system will appear to the programmer, eliminating the gap between the behavior expected by the programmer and the actual behavior supported by a system. Effectively, the consistency model places restrictions on the values that can be returned by a read in a shared-memory program execution. Each consistency model.

linearizability

- Examples

Example 1.1. Consider a counter object with a single `getAndIncrement` method. The counter's sequential behavior can be defined as a set of strings such as $[^+\ ]^+_0\ \{^+\ \}^+_1\ (^+\ )^+_2$ where $[^+$ denotes an invocation (or call) of the method and $]^+_i$ denotes the response (or return) with value i. Suppose each invocation is initiated by a different thread.

A concurrent execution may have overlapping method invocations. For example, in $(^+\ [^+\ ]^+_0\ \{^+\ \}^+_1\ )^+_2$ the execution of $(^+\ )^+_2$ overlaps with both $[^+\ ]^+_0$ and $\{^+\ \}^+_1$, whereas $[^+\ ]^+_0$ finishes executing before $\{^+\ \}^+_1$ begins. Consider the following four executions.

$(^+\ [^+\ ]^+_0\ \{^+\ \}^+_1\ )^+_2 \qquad (^+\ \{^+\ \}^+_1\ [^+\ ]^+_0\ )^+_2 \qquad [^+\ (^+\ )^+_2\ \{^+\ \}^+_1\ ]^+_0 \qquad [^+\ (^+\ )^+_2\ ]^+_0\ \{^+\ \}^+_1$

Linearizability states roughly that *every* response-to-invocation order in a concurrent execution must be consistent with the sequential specification. Thus, the first execution is linearizable, since the response of $[^+\ ]^+_0$ precedes the invocation of $\{^+\ \}^+_1$ in the specification. However, none of the other executions is linearizable. For example, the response of $\{^+\ \}^+_1$ precedes the invocation of $[^+\ ]^+_0$ in the second execution, but not in the specification.

- Guarantees: guarantees that parallel, interleaved accesses of a number of process behave as sequential ones (http://bit.ly/1ArLdjI); two linearizable executions combined also form a linearizable execution. This is an important property, since it allows us to combine linearizable components to get strongly consistent systems. (http://bit.ly/1nZ8CDI); every linearizable execution is sequentially consistent (AMP - cap3 - pg54)
- Applications: good way to describe components of large systems, where components must be implemented and verified independently (AMP - cap3 - pg55); allow parallel access to data structures like stacks, queues or sets (http://bit.ly/1ArLdjI)

quiescent consistency

- Examples: (artigo - Between Linearizability and Quiescent Consistency)

Example 1.1. Consider a counter object with a single `getAndIncrement` method. The counter's sequential behavior can be defined as a set of strings such as $[^+\ ]^+_0\ \{^+\ \}^+_1\ (^+\ )^+_2$ where $[^+$ denotes an invocation (or call) of the method and $]^+_i$ denotes the response (or return) with value i. Suppose each invocation is initiated by a different thread.

A concurrent execution may have overlapping method invocations. For example, in $(^+\ [^+\ ]^+_0\ \{^+\ \}^+_1\ )^+_2$ the execution of $(^+\ )^+_2$ overlaps with both $[^+\ ]^+_0$ and $\{^+\ \}^+_1$, whereas $[^+\ ]^+_0$ finishes executing before $\{^+\ \}^+_1$ begins. Consider the following four executions.

$(^+\ [^+\ ]^+_0\ \{^+\ \}^+_1\ )^+_2 \qquad (^+\ \{^+\ \}^+_1\ [^+\ ]^+_0\ )^+_2 \qquad [^+\ (^+\ )^+_2\ \{^+\ \}^+_1\ ]^+_0 \qquad [^+\ (^+\ )^+_2\ ]^+_0\ \{^+\ \}^+_1$

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. The first three executions above are quiescently consistent simply because there are no non-trivial quiescent points. The last execution fails to be quiescently consistent since the order from $(^+\ )^+_2$ to $\{^+\ \}^+_1$ is not preserved in the specification.

- Guarantees: quiescent consistency components can be combined to form quiescently consistent components again. (http://bit.ly/1nZ8CDI); operations of any processors separated by a period of quiescence should appear to take effect in their real-time order;
- Applications: An Elimination Tree (http://bit.ly/UA4sXy); quiescence guarantees are useful in shared memory systems where communication delays can be bounded in the absence of system failures.

sequential consistency

- Examples

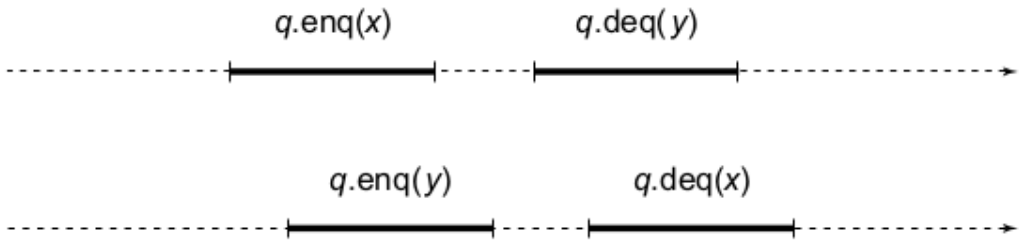

Figure 3.6 There are two possible sequential orders that can justify this execution. Both orders are consistent with the method calls' program order, and either one is enough to show the execution is sequentially consistent.

(livro AMP - capítulo 3), (pg11 - http://stanford.io/1rCtj85)

- Guarantees: sequential consistency maintains the sequential order among operations oneach processor (pg13 - http://stanford.io/1rCtj85); sequential consistency can be seen from the programmer as if the multiprocessor has no cache memories and all memory accesses go to memory, which serves one memory request at a time. (http://fileadmin.cs.lth.se/cs/Education/EDAN25/F03.pdf)
- Applications: good way to describe standalone systems, such as hardware memories, where composition is not an issue (AMP - cap3 - pg55)

serializability

- Examples: suppose the following executions:
  T1: Start; T2: Start;
  A= Read(x); B=Read(x);
  A= A+1; B=B+1;
  Write(x,A); Write(y,B);
  Commit; Commit;
  H =r1[x] r2[x] w1[x] c1 w2[y] c2
  H is equivalent to executing T2 followed by T1; note, H is not equivalent to T1 followed by T2; also note that T1 started before T2 and finished before T2, yet the effect is that T2 ran first.
- Guarantees: the result of any execution is the same as if the transactions of all the processors were executed in some sequential order.
- Applications: any transaction-based systems..

happens-before consistency

- Examples: (http://preshing.com/20130702/the-happens-before-relation/)
- Guarantees: let A and B represent operations performed by a multithreaded process. If A *happens-before* B, then the memory effects of A effectively become visible to the thread performing B before B is performed.
- Applications:

Similarities and Differences

Quiescent consistency is a compositional consistency model. A consistency model is compositional if and only if the specification of every object in a system satisfies the consistency model implies that the system as a whole satisfies the consistency model. Sequential consistency is not a compositional consistency model. Quiescent consistency is useful for systems that assume a compositional consistency model, but quiescent consistency does not preserve program order.

Linearizable consistency can be viewed as a special case of strict serializability where transactions are restricted to those consisting of a single operation applied to a single object. Serializable consistency is an extension of the sequential consistency model, where the result of any execution is the same as if the transactions of all the processors were executed in some sequential order.

Serializability is a stronger consistency model than linearizability. The serializable model allows programmers to reason about transactions as if they were sequential programs. Serializability is not a composable consistency model.

Both quiescent consistency, linearizability, and sequential consistency are non-blocking consistency models. A property is inherently blocking if it can be enforced only by blocking a transaction's data access operations until certain events occur in other transactions. serializability is not.

Serializability is about the outcome of a collection of operations/the "system" being expressible as a specific ordering ("as if execution took place in a specific order...") of all the operations. Linearizability is a property of a single subset of operations in the system... an operation/set of operations are linearizable if they appear to the other operations as if they occurred at a specific instant in (logical) time with respect to the others. 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