Requirements for a deadlock:

Priority Inversion:

Java synchronized, lock reentrancy(recursive lock),

**Process:**

IPC, process states

**Thread:**

Schedule, kernel thread, user thread

|  |  |
| --- | --- |
| Defined in header <atomic> | |
| Value | Explanation |
| memory\_order\_relaxed | Relaxed operation: there are no synchronization or ordering constraints,  only atomicity is required of this operation. |
| memory\_order\_consume | A load operation with this memory order performs a *consume operation*  on the affected memory location: no reads in the current thread dependent on the value currently loaded can be reordered before this load. This ensures that writes to data-dependent variables in other threads that release the same atomic variable are visible in the current thread. On most platforms, this affects compiler optimizations only. |
| memory\_order\_acquire | A load operation with this memory order performs the *acquire operation* on the affected memory location: no memory accesses in the current thread can be reordered before this load. This ensures that all writes in other threads that release the same atomic variable are visible in the current thread. |
| memory\_order\_release | A store operation with this memory order performs the *release operation*: no memory accesses in the current thread can be reordered after this store. This ensures that all writes in the current thread are visible in other threads that acquire or the same atomic variable and writes that carry a dependency into the atomic variable become visible in other threads that consume the same atomic. |
| memory\_order\_acq\_rel | A read-modify-write operation with this memory order is both an *acquire operation* and a *release operation*. No memory accesses in the current thread can be reordered before this load, and no memory accesses in the current thread can be reordered after this store. It is ensured that all writes in another threads that release the same atomic variable are visible before the modification and the modification is visible in other threads that acquire the same atomic variable. |
| memory\_order\_seq\_cst | Same as memory\_order\_acq\_rel, plus a single total order exists in which all threads observe all modifications (see below) in the same order. |