Lecture 16 Summary

This lecture talks about MCS locks. The paper discusses four implementations for a mutual-exclusion spin lock. Each lock can be seen as an attempt to eliminate some deficiency in the previous design. The simplest mutual exclusion lock, which can be found in all operating system textbooks and widely used in practice. It uses a polling loop to access a Boolean flag that indicates whether the lock is held. Each processor repeatedly executes a test\_and\_set instruction in an attempt to change the flag from false to true, thereby unlock the lock. The lock will be released after used. But it degraded performance not only of the memory bank in which the lock resides but also of the processor/memory interconnection network and the processor that owns the memory bank. The second one is the ticket lock, which consists of two counters. One containing the number of requests to acquire the lock, an the other the number of times the lock has been released. A processor acquires the lock by performing a fetch\_an\_increment operation on the request counter and waiting until the result is equal to the value of the release counter. It releases the lock by incrementing the release counter. But it still causes substantial memory and network contention through polling of a common location. The third one is array-based queuing locks. Each processor uses the atomic operation to obtain the address of a location on which to spin. Each processor spins on a different location and in a different cache line. The forth one is list-based queuing lock, which: guarantees FIFO ordering of lock acquisitions; spins on locally-accessible flag variables only; requires a small constant amount of space per lock; works equally well on machines with and without coherent caches. Every processor using the lock allocates a qnode record containing a queue link and a Boolean flag. Each processor has one variable during the acquire-lock operation. Processors holding or waiting for the lock are chained together by the links. Each processor spins on its own flag. Each processor in the queue holds the address of the record for the processor behind it. Compare\_and\_swap enables a processor to determine whether it is the only processor in the queue. The spin in acquire\_lock waits for the lock to become free.