EE 382C/361C: Multicore Computing **Fall 2016**

Lecture 18: November 27

Lecturer: Vijay Garg *Scribe: Huy Doan*

18.1 Atomic Scan

Scan → atomic read of multiple locations Update → atomic write of a single location

Scenario: Assume SWSR and the memory as follow

M0	0
M1	0
M2	0

Figure 18.1: Initial Memory

We are doing two operations W(M0,1), W(M1,1) sequentially. The memory states M0M1M2 we expect to observe are 100, 110. However, a scan may result in the state 010. This result means that the scan operation is not linearizable.

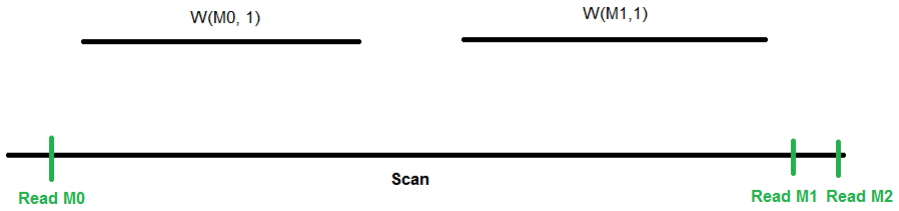

Figure 18.2: Bad Scan

Solution:
Update
Write the new value with the updated timestamp
Scan
Collect an entire array in W
Loop:
Read the entire array again to make sure that there is no change to the timestamp (2nd collect)
If there is some change, go back to loop

→ The algorithm is lock-free but not wait-free because if writes keep coming, scan has to loop and wait

Memory	Value	Timestamp
M0	1	1
M1	1	2
M2	0	0

Figure 18.3: Memory with timestamp

NOTE: It is impossible to solve the dual problem which is write to multiple locations and read a single location in a wait-free manner.

18.2 Consensus

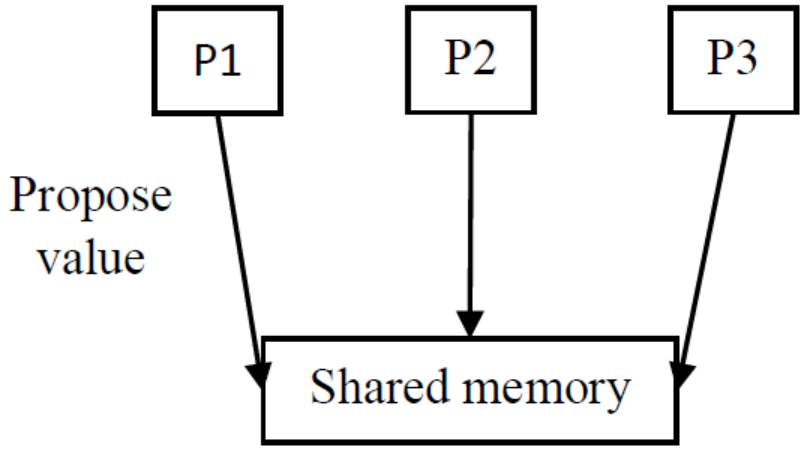

Figure 18.4: Consensus Problem

Consensus is a problem that requires a given set of processes to agree on an input value. As depicted in Figure 18.4, each process proposes its own value and the system has to decide which value among those that all processes will agree on.

The **requirements** on any object implementing consensus are as follows:

- Agreement: No two correct processes decide on different values.
- Validity: The value decided must be proposed by some process.
- Wait-freedom: Decides in a finite number of steps.

18.2.1 Bully Algorithm

Assume that we have two process P0 and P1 and no process fails. The propose values of P0 and P1 are stored in the array prop[2]. The bully algorithm solves the consensus problem by just simply picking prop[0] at all time.

Consensus 1: Bully Algorithm

```
Pi
    Write your proposal to the prop array
    Every process waits until prop[0] becomes non-null
    Decide on prop[0]
```

Obviously, this algorithm is not fair and not fault-tolerant in case that P0 fails.

18.2.2 Global State Overview

Global State: the state of the shared memory

Formally, let $G.V$ is the set of decision values reachable from global state G. We say G is **bivalent** if $|G.V| = 2$ and **univalent** if $|G.V| = 1$. In the latter case, we call G $0-valent$ if $G.V = 0$ and G $1-valent$ if $G.V = 1$. The bivalent state captures the notion of indecision.

A bivalent state is **critical** if any action by any process leads to a 0-valent or 1-valent state

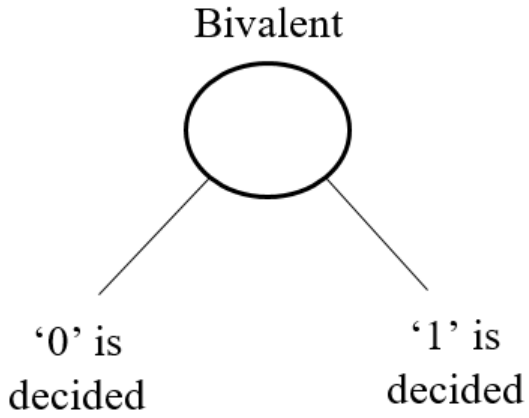

Figure 18.5: Bivalent State

Claim 1: There exists an initial bivalent global state for any consensus protocol.

Proof: Consider the most simple case where there are only two processes. If P1 is slow and P0 runs solo, P0 will eventually decide on '0'. Similarly P1 can run solo and decide on '1'. Therefore, the initial state G is bivalent.

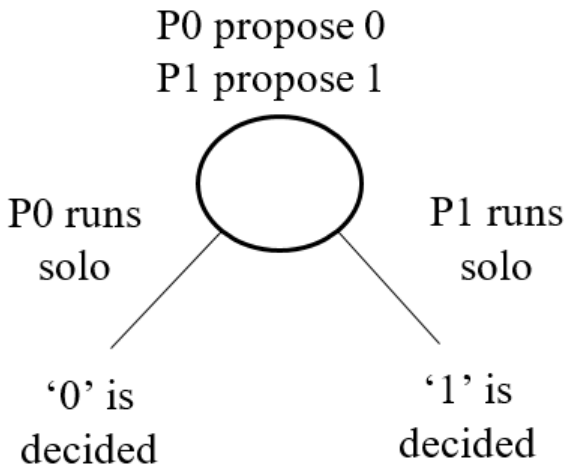

Figure 18.6: Bivalent State of Two Processes

Claim 2: There exists a critical global state for every consensus protocol.

18.2.3 Theorem

It is impossible to solve consensus with just read and write.
Proof: We show that even in a two-process system, atomic registers cannot be used to go to non-bivalent states in a consistent manner. We perform a case analysis of events that can be done by two processes, say, P and Q in a critical state S. Let e be the event at P and event f be at Q be such that e(S) has a decision value dierent from that of f(S). We now do a case analysis:

- Case 1: e and f are operations on different registers. We observe that ef(S) and fe(S) are identical and, therefore, have the same decision value. However, we assume earlier that e(S) and f(S) have different decision values, which implies that e(f(S)) and f(e(S)) have different decision values since the decision values cannot change.

- Case 2: e is read and f is write. When P does e, the state of Q does not change. Therefore, f(S) and f(e(S)) are identical and have the same decision values.

- Case 3: e and f are write on the same registers. Obviously, f(S) and f(e(S)) are identical and have the same decision values.

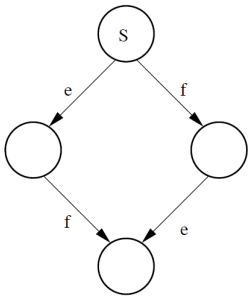

Figure 18.7: Case 1: e and fa are operations on different registers

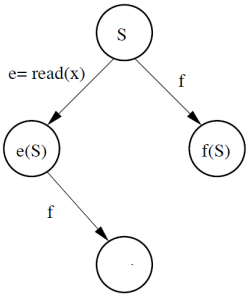

Figure 18.8: Case 2: one read one write

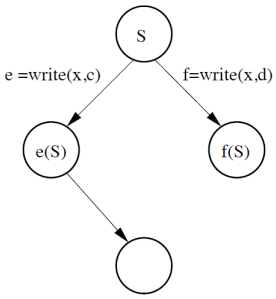

Figure 18.9: Case 3: two writes on the same register

18.2.4 Consensus Number

Consensus number of a shared object class O is the maximum number of processes that use object from class O to solve consensus

Operation	Consensus Number
R/W Register	1
TestAndSet	2
Swap	2
GetAndIncrement	2
CAS	∞

Figure 18.10: Consensus Number of Some Operations

Queue has consensus number 2 Assume a queue has two values *Win* and *Lose*. P0 and P1 can enqueue/dequeue atomically.

Consensus 2: Consensus with Queue

```
Pi
    Write proposal to the prop array
    Dequeue from Q
    If I win, choose my value
    otherwise choose the other's value
```

Theorem: There is no wait-free algorithm to build single producer - multiple consumers Queue using atomic read write registers.

CAS can be used to solve any consensus problem

Consensus 3: Consensus with CAS

```
Register R, initialized to -1
Pi
    Write my proposal to the prop array
    Do R.CAS(-1, mypid)
    if succeed
        decide prop[mypid]
    else
        decide prop[R]
```

References

[1] Vijay K. Garg, Introduction to Multicore Computing

[1] Vijay K. Garg, Concurrent and Distributed Computing in Java (2004), pp. 236