Ch5

- In a distributed system, a site can either be requesting CS execution, executing CS, or none of the above.
- Requirements for solutions:
  - Deadlock free, starvation free, Fair, Fault tolerant
- Metrics of performance (loading conditions)
  - # of messages needed for CS
  - Synch. Delay – time between one site leaving CS and another entering.
  - Response time – Time interval between CS request and end of CS
  - Throughput: rate at which system executes CS.
    - 1 / (snych. delay + CS execution time)

Solutions

- Centralized approach: Make a single site responsible for permissions.
  - Needs only 3 messages / CS (which 3 ?)
  - Single point of failure, load on central site, 2T synch. Delay
- Lamports algorithm (non token based, FIFO delivery)
  - When Si needs CS, it sends REQ(tsi, i) to all sites in its request set., and places it in its request queue. A site Sj which receives this places it in its own queue, and sends a timestamped REPLY message
  - Si can enter CS when
    - Its request is as the top of the queue
    - It has a reply from all sites it sent a message to with timestamp > timestamp of request
  - Upon exiting CS, removes its request, and sends a release message to all sites. Each receiving site dequeues the request as well

Does it work ?

- Can Prove by contradiction
  - Basically this means that a process entered CS even though a request from another process with lower timestamp was in its queue.
- Requires 3(n-1) messages / CS, sd is T
- Improvement – Ricart-Agrawala Algorithm
  - A request is sent just as in Lamport's algo.
  - On receiving a request, a reply is sent if this site is neither executing its CS nor requesting it. Otherwise, timestamps are compared and a reply sent if the received tstamp is lower than the local tstamp. Otherwise defer.
  - Enter CS when reply received from all.
  - Upon exiting CS, send replies to defered sites.
- Note that once I have clearance to go into CS, I can do so many times as long as I don't send back reply.