C.A.R. Hoare
Monitors: An Operating System Structuring Concept
Communications of the ACM 17, 10, October 1974, pp. 549-557

① Wanted to introduce a mechanism that would provide safe, modular access to shared mutable state. Writing multi-threaded programs is difficult and error-prone. Hoare sought to make this easier. ✓ ✓ ✓

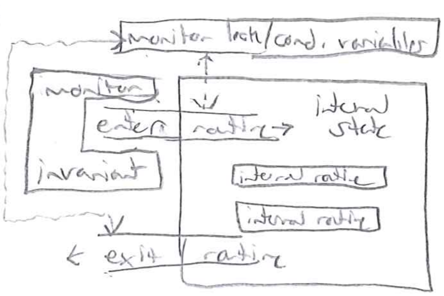
② Proposed a structure (monitor) for scheduling resources for parallel user processes. Monitor was able to protect the on-going process from interuption, and meanwhile, other processes would need to wait to enter. This paper also gives example (reader & writer) about how the schema works. ✓ ✓ ✓

③ Some what interesting: Monitors can be used to implement semaphores. This means that monitors are at least "as powerful" as semaphores. ✓

More interesting/necessary: Semaphores can be used to implement monitors. ✓

- Main binary semaphore "mutex" (init to 1)
  - P(mutex) on entry
  - V(mutex) on some exits
- "urgent" semaphore (init to 0)
  - P(urgent) when signalling condition variables in the monitor
  - V(urgent) on exit or wait if urgentcount > 0.

4)



(Okay so it's not the best picture.) ✓

⑤ Bounded buffer example

```
monitor : bb.
array[n]; last = 0, Count = 0;
procedure append(x)
    if count == N  nonfull.wait();
        buffer[last++] = x;
        count++;
        nonempty.signal
 end procedure
Procedure consume
    if count == 0  nonempty.wait();
        x = buffer[last - count]
        nonfull.signal
end procedure
```

Note : Signal should always give the mutex lock to the thread/process waiting on the corresponding C.V. [i.e. the thread/proc waiting outside on the lock cannot acquire lock]

Implication : The signalled client doesn't need to recheck the condition → ~~if~~ not while

⑥ Monitors Vs Semaphores

- Organizes shared data, code and synchronization in structured fashion.
- Separates mutual exclusion and scheduling.
- Less errors
- Parameters required, need to give invariants when monitor entered and after wait.

Stefan Savage, Michael Burrows, Greg Nelson, Patrick Sobalvarro, and Thomas Anderson
Eraser: A Dynamic Data Race Detector for Multithreaded Programs
Transactions on Computer Systems, Vol. 15, No. 4, November 1997, Pages 391–411.

✓ (1.) Finding data races using previously existing debugging tools is hard. Making synchronization errors is easy. Eraser aims to automate finding synchronization errors that could lead to data races. (via dynamic instrumentation).

2) Approach: Use the Lockset algorithm to detect races. Every shared variable is protected by some lock. Eraser checks whether program respects this discipline by monitoring reads & writes it executes from execution history. If there is an access to shared variable that is conflicting and there is no lock, then eraser identifies that as a data race.

(3) Something interesting: The Lockset algorithm can be enhanced to support reader/writer locks ~~[illegible]~~ by checking the "type" of access to the variable under consideration (read or write) and removing read-locks from the candidate set on write access.

(4) Amazing quote: "The second topic is deadlock. If the data race is Scylla, the deadlock is Charybdis."