Monitors

- Hoare/Brinch Hansen 1973:

  Monitor = data abstraction + atomicity + synchronization

Data abstraction

- Given by class construct: Private data variables + operations

Atomicity

- By implicit mutual (or R/W) exclusion among operations
- By explicit use of critical sections in operations

Synchronization

- By explicit *condition queues* ($wait$, $signal$)
  - Many variations in sematics
- By use of *guards* (**when** B)

Condition Queues

- Explicit mechanism for condition synchronization within monitors
- A condition queue is associated with a monitor, e.g. by declaration

Basic queue operations

$wait(c)$	Leave monitor and enter c *atomically*	
$signal(c)$	Wake up a single process waiting on c if any	
$signal\_all(c)$	Wake up all processes waiting on c	[SC only]

- Different semantics for signalling process:

SC	Signal-and-continue. Signaller continues in monitor
SW	Signal-and-wait. Woken process takes over monitor
Hoare	Signal-and-urgent-wait SW + signalling processes have priority to reenter.

Auxiliary Condition Queue Operations

- Given: **var** $c : condition$

Queue size

- $empty(c)$ No processes are currently waiting on c
- Queue length can be maintained in explicit monitor variables.

Priority Queuing

- $wait(c, rank)$ Wait in c according to $rank$ (ascending)
  $minrank(c)$ Return rank of first process waiting in c

Monitor Invariants

Idea

- To express local safety properties ensured by the *atomicity* of monitor operations.

Definition

- A *monitor invariant* I is a predicate on the state of a monitor M that must be true whenever the monitor is free.
- A monitor is *free* when new processes may start executing monitor operations.

Consequences

- Any process can assume I at the start of a monitor operation.
- The invariant will put *constraints* on the order of operations.
- The constraints may express properties of the whole program.

Stating Monitor Invariants

- A monitor invariant may be stated in terms of
  - The monitor variables
  - Added history variables
  - The state of condition queues
  - The state of reentry/signalling queues
- Notation:

  $waiting(c)$ Number of processes waiting on c

  $woken(c)$ Number of processes awakened from c

- Refinement, e.g.:

  $waiting_{op}(c)$ Number of processes waiting on c from operation op

Sharing via Monitor

-
```
monitor Balance
   var SUM, ITEMS : integer := 0;
   procedure ADD(w : integer)
      SUM := SUM + w
      ITEMS := ITEMS + 1
   procedure PRINT()
      if ITEMS ≠ 0 then
         print SUM/ITEMS
   procedure CLEAR()
      SUM := 0
      ITEMS := 0
end
```