Mutual Exclusion

Problem specification

- The general mutual exclusion scenario
  - Each process must be able to enter the critical section
  - Each process must be able to exit the critical section

- Safety property Mutual exclusion
  - If two or more processes try to enter the critical section simultaneously, the algorithm must ensure that at most one process enters the critical section.

- System requirements
  - Deadlocks must not occur

- Assumptions
  - Every individual base memory cell (word) load and store access is atomic
  - Processes do not delay infinitely
  - Processes are non-preemptible

- Instructions to critical sections of two or more processes must never be interleaved!

- Further assumptions
  - Pre and post conditions can be examined before and after the critical section
  - Processes do not enter a transition before a critical section
  - No more operating context in critical sections

Mutual exclusion: First attempt

```plaintext
task Body

C1 := In_CS; C2 := Out_CS;
loop
when Turn = 1;

P1; CSS := In_CS; P1;

C1 := Out_CS; C2 := Out_CS;
loop
when Turn = 1;

P1; CSS := In_CS; P1;

C1 := Out_CS; C2 := Out_CS;
end

end
```
Mutual Exclusion: Peterson's Algorithm

**Problem specification**

**Mutual exclusion:**

- Instructions from critical sections of two or more processes must never be interleaved!

**Task Range**

- **Type**
  - **Task_Range**
    - **type**
      - **mod**
        - **2**

- **Critical_Section_State**

- **CSS**
  - **this_Task**
  - **of**
    - **Task_Range**

- **Las**
  - **First**

- **CSS (this_Task)**

- **One_Of_Two_Tasks**

- **Turn**

- **G**

- **Ef**

- **Realistic hardware support**

- **Every**

- **No starvation!**

- **Critical_section**

- **G**

- **Ef**

- **Atomic**

- **P (this_id: Task_Range);**

- **Choosing (this_id) := True;**

- **id /= this_id**

- **Ticket (this_id) := Max (Ticket) + 1;**

- **Choosing (this_id) := False;**

- **Ticket (this_id) := 0;**

- **end loop**

- **P;**

- **for_leave**

- **for_enter**

- **for_loop**

- **C := 0;**

- **L := 1;**

- **C := 0;**

- **L := 1;**

- **end loop**

- **end loop**

- **end loop**

- **end loop**

- **Realistic hardware support**

- **Atomic test-and-set operations:**

- **Atomic exchange operations:**

  - **(Trip = L := C := 1)**

- **Memory cell reservations:**

  - **C := 0;**

  - **L := 1;**

  - **C := 0;**

  - **L := 1;**

- **Individual starvation possible!**

- **Busy waiting loops!**

- **G**

- **Ef**

- **Atomic instruction**

  - **No check! No global lock**

  - **No check! No global lock**

  - **No check! No global lock**

  - **No check! No global lock**

  - **No check! No global lock**

- **End of atomic memory access**

- **Beyond atomic memory access**

- **Individual instability**

  - **Other processes or devices share the reservation**

- **Works for any dynamic number of processes.**

- **mutually exclusive execution of their critical regions**

- **For two processes, 0/1/0/1/0, globally would mean unreadable (b 3/2/3/2/3)***
Semaphores

S1, S2 : Semaphore := 1;

task body Pi is
begin
loop ------ non_critical_section_i;
wait (S1);
wait (S2);
------ critical_section_i;
signal (S2);
signal (S1);
end loop;
end Pi;

task body Pj is
begin
loop ------ non_critical_section_j;
wait (S2);
wait (S1);
------ critical_section_j;
signal (S1);
signal (S2);
end loop;
end Pj;

G Mutual exclusion!
No global livelock!
G Works for any dynamic number of processes.
G Individual starvation possible!
G Deadlock possible!

Summary

Mutual Exclusion

- Definition of mutual exclusion
- Atomic load and atomic store operations
- Decker's algorithm, Peterson's algorithm
- Bakery algorithm
- Realistic hardware support
  - Atomic test-and-set, atomic exchanges, memory cell reservations
- Semaphores
  - Basic semaphore definition
  - Operating-system style semaphores