Mutual Exclusion

The general mutual exclusion scenario

- A processor contains a list of instructions
- Each instruction belongs to a critical section

Safety property: Mutual exclusion

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

Further assumptions:

- Pre and post-conditions can be executed before and after each critical section
- Each process contains exactly one critical section
- Each process contains at least one critical section

Mutual exclusion: First attempt

Turn : Task_Range := Task_Range'First;

loop

when CSS (this_Task) = Out_CS then

C1 := In_CS;

else

C2 := In_CS;

end if;

exit when C1 = Out_CS;
exit when C2 = Out_CS;

end loop;

CSS (this_Task) := Out_CS;

Mutual exclusion: Second attempt

type Critical_Section, Task_Range, Task_Token is

C1, C2: Critical_Section_State := Out_CS;

Turn : Task_Range := Task_Range'First;

loop

when CSS (this_Task) = Out_CS then

C1 := In_CS;

else

C2 := In_CS;

end if;

exit when C1 = Out_CS;
exit when C2 = Out_CS;

end loop;

CSS (this_Task) := Out_CS;

Mutual exclusion: Third attempt

type Critical_Section, Task_Range, Task_Token is

C1, C2: Critical_Section_State := Out_CS;

Turn : Task_Range := Task_Range'First;

loop

when CSS (this_Task) = Out_CS then

C1 := In_CS;

else

C2 := In_CS;

end if;

exit when C1 = Out_CS;
exit when C2 = Out_CS;

end loop;

CSS (this_Task) := Out_CS;

Mutual exclusion: Deckers Algorithm

type Critical_Section, Task_Range, Task_Token is

C1, C2: Critical_Section_State := Out_CS;

Turn : Task_Range := Task_Token'First;

loop

when CSS (this_Task) = Out_CS then

C1 := In_CS;

else

C2 := In_CS;

end if;

exit when C1 = Out_CS;
exit when C2 = Out_CS;

end loop;

CSS (this_Task) := Out_CS;

No mutual exclusion!

No deadlock!

No starvation!
The general mutual exclusion scenario

- A process can get only one permission consecutively, i.e., instructions to enter their critical sections.
- Safety property for mutual exclusion: instructions from critical sections of two or more processes must never be interleaved.

- Memory requirements:
  - Every process has multiple processes to enter their critical sections.
  - No deadlock: a process which tries to enter one of the critical sections must eventually get access to it.
  - No livelock: The process which wants to enter the critical section must be blocked frequently in a cycle and when there are no constraints.

Mutual exclusion!
No deadlock!
No global live-lock!

Mutual Exclusion: atomic test-and-set operation

L := C; C := 1

when L = 0:

L := 1; C := 0

Test and set

Does that work?
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 works too?

Mutual Exclusion

Summary

• Definition of mutual exclusion
• Atomic load and atomic store operations
  • ... some classical errors
• 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 semaphore

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