Mutual Exclusion

Mutual exclusion: First attempt

```plaintext
Turn: Task_Token := 0;
P0
begin
G := G + G;
end;
```

Assumption 1: Take the suspension of all mutual exclusion as atomic

Critical_Section_State := (others => Out_CS);

loop

if Turn = 0;

begin
non_critical_section_1;

C1 := In_CS;

end

when

Turn := Turn + 1;

end

end

Mutual Exclusion

Atomic load & store operations

Assumption 2: there is a non atomic combined radiation access

G := G + G;

Assumption 2: there is a non atomic combined radiation access

end

Turn: Task_Token := 0;
P0

end

References for this chapter


The general mutual exclusion scenario

- Suspenseful context: if the exclusion requires user identity, the connection between the tasks is context-sensitive.
- Safety property: Mutual exclusion: Instructions from critical sections of two or more processes must never be interleaved.
- More requirements:
  - No duplicates of one or multiple processes to enter their critical section at the same time
  - No deadlocks or changes of state caused by multiple processes entering the critical sections simultaneously.

- Williness: The choices which programs may enter the critical sections must be done efficiently and fairly, so that also other programs may enter the critical sections.

Mutual Exclusion

Problem specification

- Problem specification: the task of a program is non-critical.
- Further assumptions:
  - Pre- and post-protocols can be executed before and after each critical section
  - Pre-assumptions, that is, if no critical sections
  - Post-assumptions, that is, in critical sections

Mutual exclusion: Forth attempt

```plaintext
type CSS (other_Task) = Out_CS;
```

```plaintext
loop

if

CSS (other_Task) = Out_CS;

end

end

end

end

Making any progress?

Any better?

Patch
Mutual Exclusion!

No deadlock!, No global live-lock!

Does that work?

No livelock!

Mutual exclusion: Bakery Algorithm

---

Issues:

- Atomic operations: P (this_id: Task_Range);
- Choosing (id);
- Reservation exchanges:
  - Draw a new number (Pi, Pj);
  - If this_id < id, then Pi (this_id) := 0; L := 1; C := 0;
  - L := 0, C := L;
  - L := 0, C := 1;

---

Constant:

- Task_Range
- Natural range 0..1

Flag:

- Natural number range 0..1

---

Turn := Task_Range'First;

Last := Task_Range'First;

CSS (this_Task) := In_CS;

CSS (this_Task) := Out_CS;

CSS (this_Task) := Out_CS;

CSS (this_Task) := In_CS;

CSS (this_Task) := In_CS;

loop

L := C; C := 1;

when

end when

end loop

---

Count:

- L := 0; C := 0;
- L := 1, C := 0;
- L := 0, C := 1;

---

Process

- Task_Range
- Natural number range 0..1

---

\[
\begin{align*}
\text{G} & \text{Safety property 'Mutual exclusion critical sections} \\
\text{G} & \text{No deadlocks} \\
\text{G} & \text{No livelock!} \\
\text{G} & \text{Atomic operations: P (this_id: Task_Range);} \\
\text{G} & \text{Choosing (id);} \\
\text{G} & \text{Reservation exchanges:} \\
\text{G} & \text{Draw a new number (Pi, Pj);} \\
\text{G} & \text{If this_id < id, then Pi (this_id) := 0; L := 1; C := 0; L := 0, C := L; L := 0, C := 1;} \\
\text{G} & \text{Constant: Task_Range, Natural range 0..1} \\
\text{G} & \text{Flag: Natural number range 0..1} \\
\text{G} & \text{Issues: Atomic operations, Reservation exchanges, Choosing (id), Livelock, Deadlock} \\
\end{align*}
\]
Semaphores

\[ S_1, S_2 : \text{Semaphore} := 1; \]

\begin{verbatim}
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;
\end{verbatim}

\begin{verbatim}
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;
\end{verbatim}

Mutual exclusion! No global live-lock!

Mutual exclusion works for any dynamic number of processes.

Individual starvation possible!

Deadlock possible!