Solutions for Tutorial Session N°1: Mutual Exclusion

Objective: Learn how to verify a software solution for mutual exclusion against Dijkstra's four conditions.

Study the algorithmic solutions presented below to the mutual exclusion problem. Determine whether each solution is correct, and justify your answer with reference to Dijkstra's four conditions:

Dijkstra's Conditions:

To prove that a solution is correct, it is necessary to show that the four properties stated by Dijkstra are satisfied:

1 - Two processes can never be in their critical sections simultaneously.

2- If no process is in its critical section and a process wishes to enter it, that process must be able to do so within a finite amount of time.

3 - No process executing outside its critical section should block another process.

4 - All processes must share the same solution.

Solution 1 :

Common context: c : boolean; {initialized to false}	

Process P1;
Begin
...;
Repeat
while c do;
c := true;
CRITICAL SECTION;
c := false;
...;
Until false;
End;
	Process P2;
	Begin
	...;
	Repeat
	while c do;
	c := true;
	CRITICAL SECTION;
	c := false;
	...;
	Until false;
	End;

If both processes execute while c do; when c = false, they may both pass the while condition simultaneously and each set c:= true. As a result, **both enter the critical section**, violating the first condition. So this is not a correct solution for mutual exclusion.

Solution 2 :

Common context: c : boolean; {initialized to false} Turn : 1..2; {initialized to 1 or 2}

Process P1;
Begin
...;
Repeat
Turn := 2;
while (Turn ≠ 1) and (c) do;
c := true;
CRITICAL SECTION;
c := false;
...;
Until false;
End;
	Process Pi; {i = 1, 2}
	Begin
	...;
	Repeat
	Turn := 1;
	while (Turn ≠ 2) and (c) do;
	c := true;
	CRITICAL SECTION;
	c := false;
	...;
	Until false;
	End;

This solution is incorrect, it fails to ensure mutual exclusion (first condition) because both processes can enter the critical section simultaneously when c = false. The use of Turn helps coordinate turns, but because the test and assignment to c are not atomic, the algorithm is unsafe.

Solution 3 : (Dekker solution)

Common context: Flag: array [0,1] of boolean; {initialized to false} Turn: 1..2; {Turn initialized to 1 or 2}	

Process P1;
Begin
Repeat
::::
Flag[1] := true
while (Flag[2]) {
if (Turn ≠ 1) {
Flag[1] := false;
while (Turn ≠ 1) do ;
Flag[1] := true ;
}
}
CRITICAL SECTION;
Flag[1] := false;
Turn := 2;
::::
Until false;
End;
	Process Pi;
	Begin
	Repeat
	::::
	Flag[2] := true
	while (Flag[1]) {
	if (Turn ≠ 2) {
	Flag[2] := false;
	while (Turn ≠ 2) do;
	Flag[2] := true ;
	}
	}
	CRITICAL SECTION;
	Flag[2] := false;
	Turn := 1;
	::::
	Until false;
	End;

- Each process raises its flag[i] to signal its intent to enter the critical section.

- If both processes want to enter simultaneously (flag[0] = flag[1] = true):

  - The turn variable decides who goes first.

- The one whose turn it is **not** backs off (flag[i] := false) and waits.

- When the other finishes, it sets turn to the waiting process, ensuring fairness.

This is a correct solution of mutual exclusion, **the solution of Dekker (1965)**.

Solution 4 :

Common context: flag: array[1,2] of boolean; {initialized to false} turn: 1..2; {initialized to 1 or 2}	

Process P1;
Begin
… ;
flag[1] := true;
While turn ≠ 1 do {
While flag[2] do ;
turn := 1;
}
CRITICAL SECTION;
flag[1] := false;
… ;
End;
	Process P2;
	Begin
	… ;
	flag[2] := true;
	While turn ≠ 2 do {
	While flag[1] do ;
	turn := 2;
	}
	CRITICAL SECTION;
	flag[2] := false;
	… ;
	End;

To verify the 1st condition we must take into account all possible execution scenarios of P1 and P2. Consider the following scenario:

turn initialized to 2	

Process P1;
Begin
… ;
flag[1] := true; --------- (1)
While turn ≠ 1 do { ---------- (2) (8)
While flag[2] do ; ---------- (3)
turn := 1; ---------- (7)
}
CRITICAL SECTION; ---------- (9)
flag[1] := false;
… ;
End;
	Process P2;
	Begin
	… ;
	flag[2] := true; ---------- (4)
	While turn ≠ 2 do {---------- (5)
	While flag[1] do ;
	turn := 2;
	}
	CRITICAL SECTION; ---------- (6)
	flag[2] := false;
	… ;
	End;

Both processes are in their critical sections at the same time. Dijkstra's first property is therefore not ensured. So this solution is not a correct solution for mutual exclusion.