Sync Week01

Alexandre Flion - Arthur Adam

2022-12-03

Contents

1 Critical Section

1.1 Mutual exclusion

In this situation, both threads can't enter the critical section at the same time. Let's expose our thoughts:

- In the first case, both threads start at the same time. They both arrive at the same time in the while loop and they are stuck forever, showing the presence of a potential deadlock.

Line	flag[0]	flag[1]	lock[0]	lock[1]	Comment
A1	T				
B1		T			
A2					
B2					
A3					Condition verified
B3					Condition verified
A4				T	
B4			T		
A5	F				
B5		F			
A6					Condition verified, lock[1] = T
B6					Condition verified, lock[0] = T

- In our second case, thread A starts before thread B.
  - If thread B starts when thread A is at A2, we will remain in the same situation as our first case, with a deadlock.

Line	flag[0]	flag[1]	lock[0]	lock[1]	Comment
A1	T				
A2					
B1		T			
A3					Condition verified
B2					
A4				T	
B3					Condition verified
A5	F				
B4			T		
A6					Condition verified, lock[1] = T
B5		F			
A7	F				
B6					Condition verified, lock[0] = T

– If thread B starts when thread A is at A6, we will be in a situation where both threads hit the critical section but not at the same time.

Line	flag[0]	flag[1]	lock[0]	lock[1]	Comment
A1	T				
A2					
A3					Condition unsatisfied
A6					Condition unsatisfied
B1		T			
A9					Critial Section
B2			F		
A10	F				
B3					Condition unsatisfied
A11			F		
B6					Condition unsatisfied
A1	T				
B9					Critical Section
A2				F	
B10		F			
A3					Condition unsatisfied
B11				F	
A6					Condition unsatisfied
B1		T			
A9					Critical Section

– If thread B starts when thread A is at A9, A10 and A11, we will have a deadlock in the loop at line 6.

Line	flag[0]	flag[1]	lock[0]	lock[1]	Comment
A1	T				
A2					
A3					Condition unsatisfied
A6					Condition unsatisfied
A9					Critial Section
B1		T			
A10	F				
B2			F		
A11			F		
B3					Condition unsatisfied
A1	T				
B6					Condition satisfied, flag[0] = T
A2				F	
B7		F			
A3					Condition unsatisfied
B8		T			
A6					Condition satisfied, flag[1] = T
B6					Condition satisfied, flag[0] = T

Line	flag[0]	flag[1]	lock[0]	lock[1]	Comment
A1	T				
A2					
A3					Condition unsatisfied
A6					Condition unsatisfied
A9					Critial Section
A10	F				
B1		T			
A11			F		
B2			F		
A1	T				
B3					Condition satisfied
A2				T	
B4			T		
A3					Condition satisfied
B5		F			
A4				T	
B6					Condition satisfied, flag[0] = T
A5	F				
B7		F			
A6					Condition satisfied, lock[1] = T

- In our last case, thread B starts before thread A. We will meet the same situations as above.

With the scenarios we tought about, we can't say that both threads can enter the critical section at the same time.

1.2 Deadlock

The base state is all variables at false. Here is the table describing how to get to a deadlock in the given code.

Line	flag[0]	flag[1]	lock[0]	lock[1]	Comment
A1	T				
B1		T			
A2					
B2					
again A3					Condition verified
B3					Condition verified
A4				T	
B4			T		
A5	F				
B5		F			
A6					Condition verified, lock[1] = T
B6					Condition verified, lock[0] = T

At this point the program has reached a deadlock since neither locks can be set to $false$ and both loop require one of the locks to be set to $false$.

1.3 Fair implementation

Based on what we said previously (cf section 1.1 Mutual Exclusion), we explored each scenario we can think of.

This implementation can be starvation-free in 2 / 7 scenarios, where both thread run simultaneously without any deadlock.

On all others scenarios, we would have a deadlock and from there a starvation implementation.

2 Interleaving

Here is the list of steps to recreate in order to get $x = 2$:

- First thread loads x in R1
- Second thread does 99 loops, gets to $x = 99$
- First thread increments R1 and stores it into x, now $x = 1$
- Second thread loads x in S1, $S1 = 1$
- First thread finishes its loops, $x = 101$
- Second thread increments S1 and stores it into x, now $x = 2$

Needless to say, this scenario is highly improbable and would most likely never happen.

3 Synchronization

Refer to file week1_synchronization.py

4 Deadlock

Refer to file week1_deadlock.py

I could not get the simulator to work with my code, but it shouldn't matter as the deadlock is completely random and cannot be replicated using the simulator.

If you want to test the deadlock, just change the line with the randint to a value much lower, and it will be more probable for it to happen.