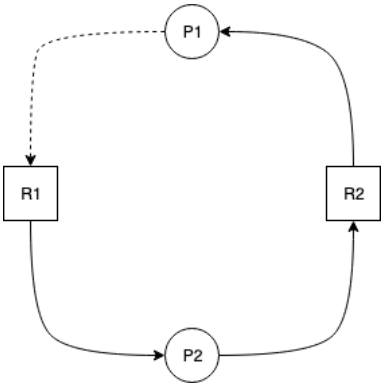
CPSC 457 – Spring 2019
Assignment 4 – Tutorial 1

Eric Austin – 30037742

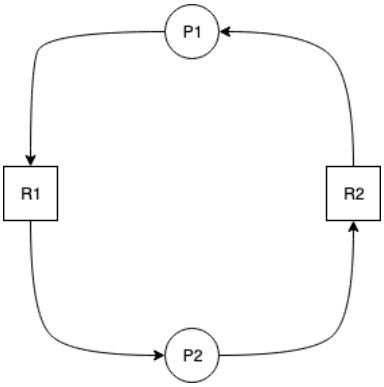
June 7, 2019

1

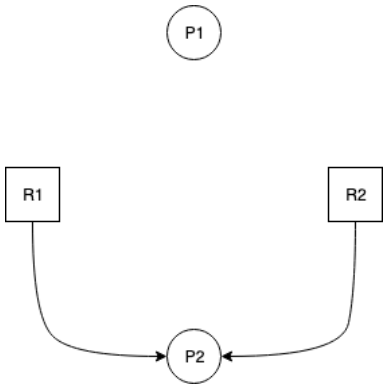
This system is in an unsafe state because a deadlock is possible if all processes claim their maximum resources. In a worse case scenario, there is no sequence of running processes that will allow them all to finish



Here we see that R1 belongs to P2, but P2 is waiting on R2 which currently belongs to P1. If P1 requests R1, which we already know it may request (ie it has a claim), then we enter a deadlock state where P1 is waiting for R1, which belongs to P2, which is waiting for R2, which belongs to P1, ...
In other words, our graph would have a cycle.

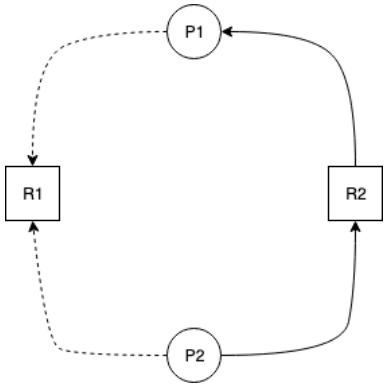


However, if P1 does not make a request for R1 then we will not have a deadlock. In this case, P1 would finish executing at which point R2 would free up and go to P2. P2 could then use R1 and R2 to finish.

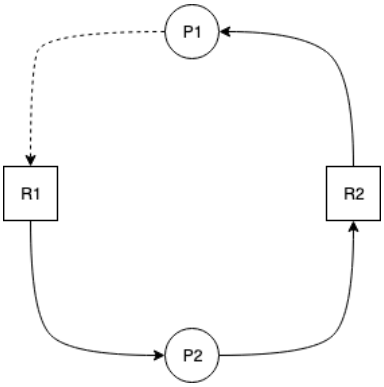


2

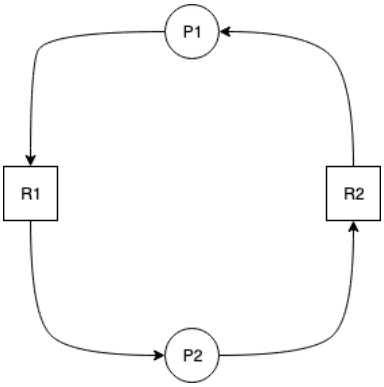
Here we can see a system that is currently in a safe state. There is a sequence of process executions that will lead to successful completion of all processes even if all processes make the maximum claim possible. However, unless the system is set up to prevent entry to an unsafe state, it is still possible to enter an unsafe state and encounter a deadlock.



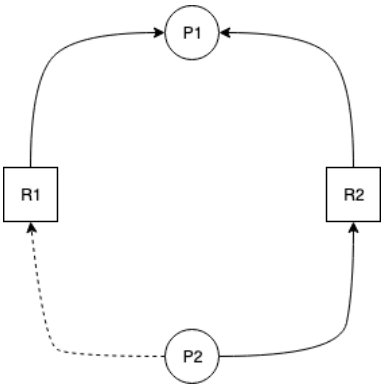
If P2 requests R1, then we enter an unsafe state (actually, the same one we saw in question 1). If the system is not prevented from entering an unsafe state, we get the same situation we saw in question 1, with R1 belonging to P2 and R2 belonging to P1, with a request for R2 from P2.



If P1 requests R1, we now find ourselves in a deadlocked state.



However, it is possible for both processes to finish from our original safe state. If P1 requests R1, then we will not get a deadlock.



Now P1 has both resources it needs to finish. Once it is completed, R2 is free for P2 to finish (and R1 is available as well if P2 decides it needs that too).

3

Process	Allocation	Request	Available
P1	0 0 1	1 2 1	0 1 0
P2	0 1 0	0 1 1	
P3	1 1 2	1 2 2	
P4	1 0 0	0 2 1	
P5	0 0 1	0 1 0	

We can grant P5's request, allowing it to finish and freeing up it's allocation of 0 1 1.

Process	Allocation	Request	Available
P1	0 0 1	1 2 1	0 1 1
P2	0 1 0	0 1 1	
P3	1 1 2	1 2 2	
P4	1 0 0	0 2 1	
P5	0 0 0	0 0 0	

We are now able to grant P2's request of 0 1 1, allowing it to finish and freeing up 0 2 1.

Process	Allocation	Request	Available
P1	0 0 1	1 2 1	0 2 1
P2	0 0 0	0 0 0	
P3	1 1 2	1 2 2	
P4	1 0 0	0 2 1	
P5	0 0 0	0 0 0	

Now we have enough available to grant P4's request, allowing it to finish and making 1 2 1 available.

Process	Allocation	Request	Available
P1	0 0 1	1 2 1	1 2 1
P2	0 0 0	0 0 0	
P3	1 1 2	1 2 2	
P4	0 0 0	0 0 0	
P5	0 0 0	0 0 0	

We can now grant P1's request, allowing it to finish and freeing 1 2 2.

Process	Allocation	Request	Available
P1	0 0 0	0 0 0	1 2 2
P2	0 0 0	0 0 0	
P3	1 1 2	1 2 2	
P4	0 0 0	0 0 0	
P5	0 0 0	0 0 0	

This is enough to grant P3's request and all processes are now finished. Thus we were not in a deadlock.

Process	Allocation	Request	Available
P1	0 0 0	0 0 0	2 3 4
P2	0 0 0	0 0 0	
P3	0 0 0	0 0 0	
P4	0 0 0	0 0 0	
P5	0 0 0	0 0 0	