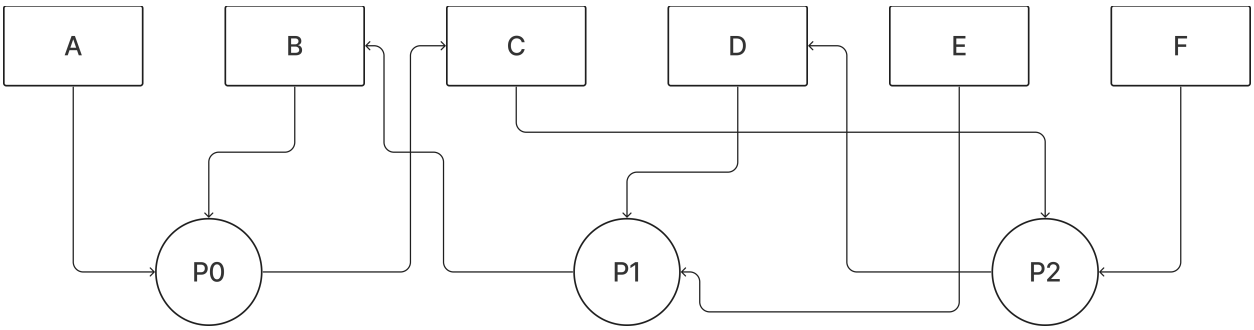
Homework 4

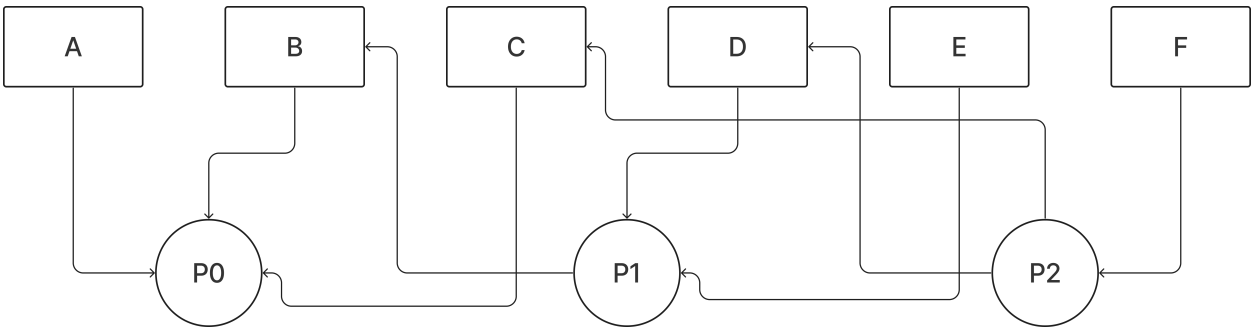
Type	Homework
Materials	
Reviewed	☐
Date	
Topic	

1a



There is a cycle in this graph, meaning there is a possibility of deadlock. P0 is holding B, and P0 is requesting C. C is held by P2, and P2 is requesting D. P1 is holding D, and P1 is requesting B. This is held by P0. Expressed differently: P0 → C → P2 → D → P1 → B → P0. This shows that this is a circular wait. Provided that there is no preemption, there is hold and wait, and that resources are mutually exclusive, then there is deadlock.

1b



We can change the orders to the following:

```
void P0()
{
  while(true) {
    get(C);
    get(B);
    get(A);
    ...
  }
}
```

```
void P1()
{
  while(true) {
    get(E);
    get(D);
    get(B);
    ...
  }
}
```

```
void P2()
{
  while(true) {
    get(F);
    get(C);
    get(D);
    ...
  }
}
```

From the graph above, we can see that there is no cycle. P0 isn't waiting on any resources, so it can finish immediately and release its resources. This allows P1 to `get` B, and it can now run the critical section, ending it by releasing its resources. Now, P2 is able to use any resource that it needs, as all are available, and it can complete. The order is therefore free of circular wait and free of deadlock.

2

	S	R	foo	bar
t=0	1	1	-	-
t=1	0	0	semWait(S) S ≠ 0, so S=S-1	semWait(R) S ≠ 0, so S=S-1
t=2	0	0	semWait(R) R == 0, so wait	semWait(S) S == 0, so wait

If both processes execute at exactly the same speed, both processes will be waiting indefinitely at the second `semWait`, as each is waiting for a semaphore to increase. Therefore, there is a chance that both are blocked forever.

3

Preventing deadlock means adopting a policy that eliminates one of the conditions for deadlock. Avoiding deadlock means making appropriate **dynamic** choices based on the current state of resource allocation. Detecting deadlock means attempting to detect the presence of deadlock and taking action to recover from it.

4

	P0	P1	P2	Available
t=0	0	-	-	4
t=1	1 get → success	1 get → success	1 get → success	1
t=2	2 get → success critical section	1 get → wait	1 get → wait	0
t=3	0 release resources	1 get → wait	1 get → wait	2
t=4	0	2 get → success critical section	2 get → success critical section	0
t=5	0	0 release resources	0 release resources	4

In the worst-case scenario, all processes request 2 resources at the exact same time. 1 of the resources is given to each, meaning we have one left over. Two of the processes would therefore wait, and one process would get that resource, allowing it to use that resource. Once that process has completed its critical section, it releases its 2 resources, allowing the other two to each get one and complete their critical section. This is demonstrated in the above table.