COMP 2432 Operating Systems
Tutorial 12

1. **Andes Trains Solution 1.**
Consider the first solution of the shared railway section problem for Andes Trains. The solution could be translated into a *concurrent program* with 2 processes as below. *Show* a possible *sequence of events* (with statement number) that have happened to cause these incidents: (*a*) a train being blocked forever, (*b*) collision of the trains. Note that there can be many possible answers.

`Initially, flag = false`	
$P_{Bolivia}$	P_{Peru}

`while (true) do {`
`1: while (flag) do;`
`2: flag = true;`
`3: < critical section >`
`4: flag = false;`
`5: < remainder section >`
`}`
	`while (true) do {`
	`1: while (flag) do;`
	`2: flag = true;`
	`3: < critical section >`
	`4: flag = false;`
	`5: < remainder section >`
	`}`

(*a*)

Time	$P_{Bolivia}$	P_{Peru}	*flag*
0			false
1	1		
2	2		true
3		1	
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			

(*b*)

Time	$P_{Bolivia}$	P_{Peru}	*flag*
0			false
1		1	
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			

2. **Andes Trains Solution 2.**
Consider the second solution proposed by the PolyU student. *Express* it as a *concurrent program*. *Show* a sequence of possible events in the context of your program that have occurred in a scenario that the Peruvian train has to wait for the Bolvian train to pass before it can take a second turn to pass the shared track.

`Initially, flag = false`	
$P_{Bolivia}$	P_{Peru}

`while (true) do {`
`1:`
`2:`
`3:`
`4:`
`5:`
`}`
	`while (true) do {`
	`1:`
	`2:`
	`3:`
	`4:`
	`5:`
	`}`

Time	$P_{Bolivia}$	P_{Peru}	*flag*	Time	$P_{Bolivia}$	P_{Peru}	*flag*	Time	$P_{Bolivia}$	P_{Peru}	*flag*
0			false	10				20			
1				11				21			
2				12				22			
3				13				23			
4				14				24			
5				15				25			
6				16				26			
7				17				27			
8				18				28			
9				19				29			

3. Andes Trains Solution 3.

Consider the third solution using two bowls. *Complete* the *concurrent program* for the Peruvian train. *Show* a sequence of possible events on (*a*) how the Peruvian train can run *twice* a day and the Bolvian train can run *once* a day, (*b*) how a *livelock* has occurred that both trains are blocked.

```
Initially, flag[0] = false and flag[1] = false
```

$P_{Bolivia}$

```
while (true) do {
1: while (true) do {
2:     flag[0] = true;
3:     if (flag[1] == false) then
4:             break;
5:     else flag[0] = false;
6: }
7: < critical section >
8: flag[0] = false;
9: < remainder section >
}
```

P_{Peru}

```
while (true) do {
1:
2:
3:
4:
5:
6:
7:
8:
9:
}
```

(*a*)

Time	$P_{Bolivia}$	P_{Peru}	*flag*[0]	*flag*[1]
0			false	false
1	1			
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				
22				
23				
24				
25				
26				
27				
28				
29				
30				

(*b*)

Time	$P_{Bolivia}$	P_{Peru}	*flag*[0]	*flag*[1]
0			false	false
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				
22				
23				
24				
25				
26				
27				
28				
29				
30				

4. **Critical Section Problem.**

Consider McDull's modification to Peterson's algorithm to the critical section problem for two processes P_1 and P_2, by making processes more "selfish". Initially, both *flag*[1] and *flag*[2] are *false* and *turn* = 1.

Program for P_1	Program for P_2

`while (true) do {`
`1: flag[1] = true;`
`2: turn = 1;`
`3: while (flag[2] and turn==2) do { };`
`4: < critical section >`
`5: flag[1] = false;`
`6: < remainder section >`
`}`
	`while (true) do {`
	`1: flag[2] = true;`
	`2: turn = 2;`
	`3: while (flag[1] and turn==1) do { };`
	`4: < critical section >`
	`5: flag[2] = false;`
	`6: < remainder section >`
	`}`

McMug points out that the new solution proposed by McDull is incorrect with respect to *mutual exclusion* and *bounded waiting*, but correct with respect to *progress*. Give a scenario with a sequence of events to show how the *mutual exclusion* property is violated. Give another scenario with a sequence of events to show how the *bounded waiting* property is violated. Give a final sequence of events as examples that the *progress* property still holds.

ME

Time	P_1	P_2	*flag*[1]	*flag*[2]	*turn*	Time	P_1	P_2	*flag*[1]	*flag*[2]	*turn*
0			false	false	1	8					
1						9					
2						10					
3						11					
4						12					
5						13					
6						14					
7						15					

PG

Time	P_1	P_2	*flag*[1]	*flag*[2]	*turn*	Time	P_1	P_2	*flag*[1]	*flag*[2]	*turn*
0			false	false	1	13					
1						14					
2						15					
3						16					
4						17					
5						18					
6						19					
7						20					
8						21					
9						22					
10						23					
11						24					
12						25					

BW

Time	P_1	P_2	*flag*[1]	*flag*[2]	*turn*	Time	P_1	P_2	*flag*[1]	*flag*[2]	*turn*
0			false	false	1	10					
1						14					
2						15					
3						16					
4						17					
5						18					
6						19					
7						20					
8						21					
9						22					
10						23					
11						24					
12						25					