

## COS214 Tutorial 9

*Roberto Togneri, 2000*

### **SOLUTIONS**

1. (a) Imagine there's a twin track railway line with a single track section going through a tunnel. How would you design a system that would ensure that only one train at a time enters the tunnel in any one direction? What are the attributes of your solution? For example, does this allow trains from one direction through even if there are no trains from the other direction?
- (b) In a restaurant there is a seating area for patrons with a narrow corridor (with a swing door at each end) for the waiters to gain access to the kitchens. It is so narrow that it will only allow for a waiter to go through it in one direction at a time. The manager installed a solution to the problem of collisions - a switch at each end that put a light on at the other end. If the light was on then you didn't enter the corridor. If the light was off, then you threw the switch, travelled through and then threw the switch at the other end to turn the light off. This worked fine, but yet there were still complaints with occasional accidents. The waiters complained that the system was unfair. Can you explain why there were still accidents and why the waiters complained?

**Answer**

(a)  
Place an *entry detector* and an *exit detector* at each end of the tunnel with one pair for each track (i.e. 4 detectors). First carriage that hits *enter detector* trips a warning light at the other end and increments a train count register. When last carriage of the train passes the *exit detector* the train register count is decremented. If the count is zero then the warning light goes off and any opposite going train is allowed onto the single track. A similar system occurs for the trains going the other way. This solution will provide minimal interference with a series of trains entering the track as the count will keep track of them and wait until they are all through. To avoid collisions and possible starvation a decision program must be run to prevent two trains from entering simultaneously (crash!) and trains entering from the same direction after one another when an opposing train is waiting (opposing train may *starve*).

(b)  
*Unfair?* If there is a queue of waiters waiting to go through then they can keep hitting the light switch and go through thereby preventing any waiters entering from the other end.  
*Problems?* Waiters hit the light simultaneously or so quickly that they don't realise what has happened and enter the corridor at the same time leading to a collision.  
*Also:* Since the light switch is a single action event then only one waiter can be permitted to travel through the corridor at any one time. If there is a queue of waiters wanting to go in the same direction it would make more sense to have them go together (but how would you do this?).

2. (a) What is meant by the term concurrent processing?
- (b) Give a definition of mutual exclusion. Give examples of where mutual exclusion is required for proper operation.
- (c) What is a race condition? What is a critical region?
- (d) What is meant by the term "deadlock"?

**Answer**

- (a) *Concurrent processing:* Processes run in parallel or concurrently.
- (b) *Mutual exclusion:* allow only one process to use a shared resource at any one time. Processes must finish using the resource before other processes are allowed access. E.g. printer, system email boxes.
- (c) *Race condition:* condition where processes are reading and writing some shared data and the final result depends on who runs precisely when. How about if processes are only reading? One reading the other writing? Both writing?  
*Critical region:* Part of a program where the shared memory is accessed. A shared file can be considered a special case of shared memory.
- (d) *Deadlock:* process A enters critical region and sets up file lock/semaphore to prevent other processes from entering. Process A blocks or dies whilst in critical region without releasing the file lock/semaphore. Therefore, other processes will indefinitely block when attempting to enter critical region => deadlock.

3. For the following program fragments *assume that ProcessNumber is a shared integer variable initialised to 1; P1Inside, P2Inside* are shared Boolean variables initialised to false; and *P1WantsToEnter* and *P2WantsToEnter* are shared Boolean variables initialised to false. Further, assume each process is in some repeating loop and so continually, but at possibly differing rates, re-executes the code shown. For each of the three cases explain why they are unsatisfactory as solutions to the mutual exclusion problem.

Process One	Process Two
(a) while(ProcessNumber==2); CriticalSection(); ProcessNumber=2; ...	(a) while(ProcessNumber==1); CriticalSection(); ProcessNumber=1; ...
(b) while(P2Inside); P1Inside=TRUE; CriticalSection(); P1Inside=FALSE; ...	(b) while(P1Inside); P2Inside=TRUE; CriticalSection(); P2Inside=FALSE; ...
(c) P1WantsToEnter=TRUE; while(P2WantsToEnter); CriticalSection(); P1WantsToEnter=FALSE; ...	(c) P2WantsToEnter=TRUE; while(P1WantsToEnter); CriticalSection(); P2WantsToEnter=FALSE; ...

**Answer**

- (a) Processes MUST alternate. Big problem if one process takes considerably longer than the other to return to the critical section.
- (b) If both processes execute the initial while loop simultaneously or process one sets P1Inside=TRUE just after process two has tested P1Inside then processes enter the critical region together → race condition.
- (c) Same as (b) except that processes will block → deadlock.

4. You are designing a mutual exclusion primitive for a computer that does not have a test and set instruction. Instead it has an instruction that interchanges the contents of a register with a memory location in a single atomic action. Use this instruction to create primitives to enter and leave critical regions.

```

Answer
MOV A,0           ; initialise memory location to 0
XMEM A
...
enter_region:
    MOV A,1
    XMEM A
    CPI 00H
    JNZ enter_region
...
leave_region:
    MOV A,0
    XMEM A

```

5. From the earlier tutorial that gave the problem of the waiters in a restaurant colliding in a one way tunnel between the eating area and the kitchen, can you now think of how to solve this problem using a simple semaphore solution? Describe any limitations of your solution.

```

Answer
Simple semaphore solution:
semaphore corridor=1;

void waiter(void)
{
while(working) {
    process_dining( );
    down(&corridor);
    enter_corridor( );
    up(&corridor);
    process_kitchen( );
    down(&corridor);
    enter_corridor( );
    up(&corridor);
}
}

Limitations? Only one waiter can be in the tunnel at any one time even if all are going in the same direction

```

6. A road bridge has only a single lane. To the North and South are feeder roads that are two laned. More than one car can be on the bridge if travelling in the same direction. Using semaphores, write the pseudo-code routines *enter\_south()*, *leave\_south()*, *enter\_north()*, *leave\_north()* that arrange for cars to safely cross the bridge [**Hint:** Readers & Writers semaphore solution]. What are the attributes and problems of your proposed solution?

```

Answer

semaphore mutexS = mutexN = 1;
semaphore bridge = 1;
int south = north = 0;

enter_south( ){
    down(&mutexS);
    south = south + 1;
    if (south == 1) down (&bridge);
    up(&mutexS);
}

leave_south( ){
    down(&mutexS);
    south = south - 1;
    if (south == 0) up (&bridge);
    up(&mutexS);
}

enter_north( ){
    down(&mutexN);
    north = north + 1;
    if (north == 1) down (&bridge);
    up(&mutexN);
}

leave_north( ){
    down(&mutexN);
    north = north - 1;
    if (north == 0) up (&bridge);
    up(&mutexN);
}

Features? Allows more than one car in the same direction to travel on bridge, but without controlled alternation (i.e. cars waiting from the opposite side may be starved if there is always at least one car coming from the opposite direction).

```