---

1) This code will definitely **NOT** work! In the simplest scenario we have iZoneOld and iZoneNew having the same value (lets say 0). Also to make things even simpler, lets going to ignore the daylight savings time.
The vSetTimeZone function becomes something like:

```
void vSetTimeZone(int iZoneOld, int iZonenew)
{
      int iHoursTemp;
      disable();
      iHoursTemp=iHours;
      enable();

      //this is a special line
      ihoursTemp= iHoursTemp + 0 + 0;

      disable();
      iHours=iHoursTemp;
      enable();
}
```

Lets say the time is 16:59:59, when we enter the vSetTimeZone function, and an interrupt happens when we are in the comment that says "this is a special line". This interrupt which happens every second will update the global variables that store time. So, iHoursTemp=16 but iHours will be 17, which is the correct time. When, when we enter the next critical section, iHours will now be rewritten with the value 16, which is very wrong. So while the code will work most of the time, there is a situation when we may have an incorrect hour in our system.

2) This particular worst case interrupt latency is : (the time it takes to finish the random piece of code) + (the time it takes to finish the interprocessor interrupt) + (the time it takes to finish the network interrupt). So it should be 250µsec + 350µsec + 100µsec.

3) A buffer is used to temporarily store data that is arriving until a program can get a chance to process the data. This allows the program to perform other tasks as the data is arriving. (note that there can also be output buffers: the program places a set of bytes into a buffer to be sent out, but does not have to wait for them to leave).

4) The shared data problem can occur when; two parts of a program can access a common data structure, and at least one of these parts (e.g., an ISR) can interrupt the other at any time. The problem occurs because both parts of the program assume that the data structure does not change while it is executing. However, when one can interrupt the other, this assumption no longer holds.

5) While we are querying for an event to happen, we are stalling other processes.

6) We should try to make interrupts execute as fast as possible, so they don't affect the throughput of our program. If we are waiting for data to arrive inside an interrupt we are definitely making the interrupt take longer to execute than it has to. One good way is to solve these type of problems is to have an interrupt that notifies the arrival of new I/O data.