**CS4023** **Week11 Tutorial Solutions**

The following questions are taken from the exercises at the end of Chapter 6 of SGG (ed. 8)

1. Explain why disabling interrupts in order to enforce critical-section execution or implement other synchronization primitives is not a good idea on a multi-processor system. (**Q6.14**)
   **Answer:** Interrupts are not sufficient in multiprocessor systems since disabling interrupts only prevents other processes from executing on the processor in which interrupts were disabled; there are no limitations on what processes could be executing on other processors and therefore the process disabling interrupts cannot guarantee mutually exclusive access to program state.

2. Explain why disabling interrupts on a single-processor system doesn't work, either. (**Q6.13**)
   **Answer:** If a user-level program is given the ability to disable interrupts, then it can disable the timer interrupt and prevent context switching from taking place, thereby allowing it to use the processor without letting other processes execute.

   Also, the system clock is updated at every clock interrupt. If interrupts were disabled – particularly for a long period of time – it is possible the system clock could easily lose the correct time. The system clock is also used for scheduling purposes. For example, the time quantum for a process is expressed as a number of clock ticks. At every clock interrupt, the scheduler determines if the time quantum for the currently running process has expired. If clock interrupts were disabled, the scheduler could not accurately assign time quantums. This effect can be minimized by disabling clock interrupts for only very short periods.

3. Argue carefully why Dekker's Algorithm satisfies all three requirements for the critical-section problem. (**Q6.9**)
   **Answer:** This algorithm satisfies the three conditions of mutual exclusion. (1) Mutual exclusion is ensured through the use of the flag and turn variables. If both processes set their flag to true, only one will succeed, namely, the process whose turn it is. The waiting process can only enter its critical section when the other process updates the value of turn. (2) Progress is provided, again through the flag and turn variables. This algorithm does not provide strict alternation. Rather, if a process wishes to access their critical section, it can set their flag variable to true and enter their critical section. It sets turn to the value of the other process only upon exiting its critical section. If this process wishes to enter its critical section again – before the other process – it repeats the process of entering its critical section and setting turn to the other process upon exiting. (3) Bounded waiting is preserved through the use of the `turn` variable:

assume two processes wish to enter their respective critical sections. They both set their value of flag to true; however, only the thread whose turn it is can proceed; the other thread waits. If bounded waiting were not preserved, it would therefore be possible that the waiting process would have to wait indefinitely while the first process repeatedly entered—and exited—its critical section. However, Dekker's algorithm has a process set the value of turn to the other process, thereby ensuring that the other process will enter its critical section next.

4. Servers can be designed to limit the number of open connections. For example, a server may wish to have only $N$ socket connections at any point in time. As soon as $N$ connections are made, the server will not accept another incoming connection until an existing connection is released. Explain how semaphores can be used by a server to limit the number of concurrent connections. (**Q6.17**)

   **Answer** A semaphore is initialized to the number of allowable open socket connections. When a connection is accepted, the `acquire()` method is called; when a connection is released, the `release()` method is called. If the system reaches the number of allowable socket connections, subsequent calls to `acquire()` will block until an existing connection is terminated and the release method is invoked.

5. Show that, if the `wait()` and `signal()` semaphore operations are not executed atomically – and thus implemented using some type of critical section mechanism – then mutual exclusion may be violated. (**Q6.18**)

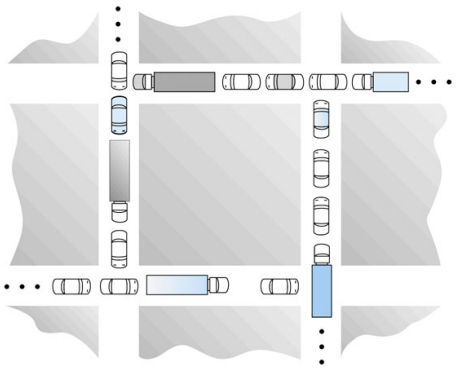
   **Answer** A `wait()` operation atomically decrements the value associated with a semaphore. If two `wait()` operations are executed on a semaphore when its value is 1, if the two operations are not performed atomically, then it is possible that both operations might proceed to decrement the semaphore value, thereby violating mutual exclusion.

6. Race conditions are possible in many computer systems. Consider a banking system with two functions: `deposit(amount)` and `withdraw(amount)`. These two functions are passed the amount that is to be deposited or withdrawn from a bank account. Assume a shared bank account exists between a husband and wife and concurrently the husband calls the `withdraw()` and the wife calls `deposit()`. Describe how a race condition is possible and what might be done to prevent the race condition form occurring. (**Q6.8**)

   **Answer** Assume the balance in the account is 250.00 and the husband calls `withdraw(50)` and the wife calls `deposit(100)`. Obviously the correct value should be 300.00. Since these two transactions will be serialized,

the local value of balance for the husband becomes 200.00, but before he can commit the transaction, the `deposit(100)` operation takes place and updates the shared value of balance to 300.00. We then switch back to the husband and the value of the shared balance is set to 200.00 - obviously an incorrect value.

7. For the traffic gridlock shown below, argue that the four necessary conditions for deadlock hold; give a simple rule for avoiding deadlocks in a system like this; and, most usefully, propose how Irish drivers can be encouraged to adhere to this rule. (**Q7.10**)



**Answer**

(a) The four necessary conditions for a deadlock are (1) mutual exclusion; (2) hold-and-wait; (3) no preemption; and (4) circular wait. The mutual exclusion condition holds since only one car can occupy a space in the roadway. Hold-and-wait occurs where a car holds onto its place in the roadway while it waits to advance in the roadway. A car cannot be removed (i.e. preempted) from its position in the roadway. Lastly, there is indeed a circular wait as each car is waiting for a subsequent car to advance. The circular wait condition is also easily observed from the graphic.

(b) A simple rule that would avoid this traffic deadlock is that a car may not advance into an intersection if it is clear it will not be able immediately to clear the intersection.