Chapter 6

23. If a process has m resources it can finish and cannot be involved in a deadlock. Therefore, the worst case is where every process has m − 1 resources and needs another one. If there is one resource left over, one process can finish and release all its resources, letting the rest finish too. Therefore the condition for avoiding deadlock is r $\geqslant$ p(m − 1) + 1.

26. The needs matrix is as follows:

01001
02100
10300
00111

If x is 0, we have a deadlock immediately. If x is 1, process D can run to completion. When it is finished, the available vector is 1 1 2 2 1 and A can run. After it finishes and returns its resources the available vector is 2 1 4 3 2, which will allow C to run and complete, the available vector is 3 2 4 4 2 and then B to run and complete. Therefore, the smallest value of x that avoids a deadlock is 1.

31. Change the semantics of requesting a new resource as follows. If a process asks for a new resource and it is available, it gets the resource and keeps what it already has. If the new resource is not available, all existing resources are released. With this scenario, deadlock is impossible and there is no danger that the new resource is acquired but existing ones lost. Of course, the process only works if releasing a resource is possible (you can release a scanner between pages or a CD recorder between CDs).

34. A deadlock occurs when a set of processes are blocked waiting for an event that only some other process in the set can cause. On the other hand, processes in a livelock are not blocked. Instead, they continue to execute checking for a condition to become true that will never become true. Thus, in addition to the resources they are holding, processes in livelock continue to consume precious CPU time. Finally, starvation of a process occurs because of the presence of other processes as well as a stream of new incoming processes that end up with higher priority that the process being starved. Unlike deadlock or livelock, starvation can terminate on its own, e.g. when existing processes with higher priority terminate and no new processes with higher priority arrive.