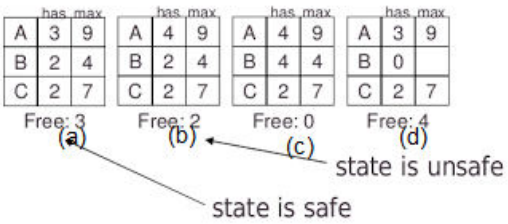
Question1



Take example of unsafe state from book “Modern Operating Systems” (p.451). We have three processes A, B, C. In (a) we have a state in which process *A* has three instances of the resource but may need as many as nine eventually, B currently has two and may need four altogether, later. Similarly, C also has two but may need an additional five. A total of 10 instances of the resource exist, so with seven resources already allocated, there are three still free. Eventually, *B* completes and we get the situation of (d). At this point we are stuck. We only have four instances of the resource free, and each of the active processes needs five. There is no sequence that guarantees completion. Thus the allocation decision that moved the system from (a) to the (b) went from a safe state to an unsafe state. Running *A* or C next starting at (b) does not work either. In retrospect, A's request should not have been granted.

It is worth noting that an unsafe state is not a deadlocked state. Starting at (b), the system can run for a while. In fact, one process can even complete. Furthermore, it is possible that *A* might release a resource before asking for any more, allowing C to complete and avoiding deadlock altogether. Thus the difference between a safe state and an unsafe state is that from a safe state the system can guarantee that all processes will finish; from an unsafe state, no such guarantee can be given.

Question2

If pages can be shared, yes. It is clearly more efficient to share the pages, to avoid having two copies of the same page in memory at the same time. One problem is that not all pages are sharable. In particular, pages that are read-only, such as program text, can be shared, but data pages cannot. For example, if two users of a timesharing system are running the same editor at the same time, and the program text is shared rather than copied, some of those pages may be in each user's working set at the same time.

Question3

General virtual memory support is not needed when the memory requirements of all applications are well known and controlled. Some examples are special-purpose processors (e.g., network processors), embedded processors, and super-computers (e.g., airplane wing design). In these situations, we should always consider the possibility of using more real memory. If the operating system did not have to support virtual memory, the code would be much simpler and smaller. On the other hand, some ideas from virtual memory may still be profitably exploited, although with different design requirements. For example, program/thread isolation might be paging to flash memory.

Question4

In order to calculate the average and median of the sizes of executable binary files on my computer, I need to take on a UNIX system at all executable files in /bin,/usr/bin libraries. When at all files need to sort and count files in order to find median size and need to sum in order to find average size. The program need to check that file from the library must be in type executable. After all program computation I received:

Median = 11588 bytes Average = 71062.7 bytes

Determine the optimal page on my computer will help formula on page 220 (“Modern Operating Systems”):

S = Average

E = 4 bytes – the size of a page table entry

P = sqrt(2 x S x E) ≈ 754 bytes - the optimal page size