1 a) In most common page table implementations the data structure is indexed by the virtual page number (VPN) so increasing the number of virtual address bits without increasing the page size will increase the size of the page table. More entries and more bytes will be needed to store the mappings for the increased number of virtual pages.

Typically the page table entry does not include the virtual address or VPN (since those are used as an index into the table) so the page table entry can stay the same size even though the virtual address gets bigger.

Most TLBs end up including both the virtual page number and physical page number so it will get bigger when the virtual address gets bigger. If you view the TLB as a cache of page table entries, the VPN can be viewed as the tag that is matched on.

1 b) Increasing the number of physical address bits does not increase the number of entries in the page table but it does increase the size of each entry. Hence the page table gets larger in size but not in entries. The page table entry normally contains the physical page number so increasing its size will increase the page table entry. The TLB includes the physical page numbers so it will increase in size.

1 c) Increasing the number of active processes increases the number of page tables but each table will stay the same size.

The page table entries will stay the same size for the same reason.

In order for the TLB to hold a increased number of active processes, the PID field use to tag processes' entries in the TLB will need to be increased resulting in a slight increase in the TLB size.

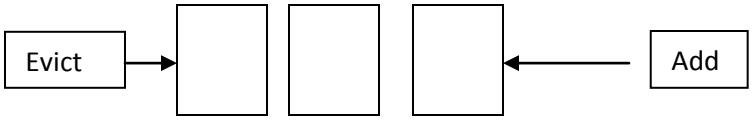
2 a) The number of code pages read from the disk should decrease. Since the code from dynamically linked libraries will be shared by many processes, it will only need to be read from disk once instead of once per process.

2 b) The number of code pages in virtual memory across all the processes should increase. Statically linking only pulls in the functions the process needs. Because the library functions are packed with the rest of the code, this often does not increase the number of pages the process uses. Dynamic linking pulls the entire library into memory and puts them in their own pages in memory.

2 c) The number of TLB misses a process suffers should increase. As mentioned in part b, because the library code is no longer packed in the same pages as the process's code the process will need to have more mapped pages, which increases the chance that a TLB miss will occur.

3. Periodically scan all resident pages of a process. If the use bit is not set, add to the page's idle time the amount of CPU time the process has used since the last scan. If the use bit is set, set the page's idle time to 0 and the use bit to 0.

4) FIFO page replacement policy means "throw/evict" out the oldest page
(oldest in terms of arrival time and not usage per say)



Mechanism used by VMS and Windows NT to implement a FIFO page replacement policy yet get behavior similar to a LRU algorithm

Instead of simply evicting a page, once eviction is done by the main FIFO queue, the page is kept in another list. Lets call it "free list". The idea is to not throw away recently evicted pages so that in case a reference comes for them in a near future then instead of again reading the entire page from memory, now we would be able to save a memory read and give the page from the free list.

So if a page is not found in the main FIFO queue then the page is first looked for in this "free list" and if the page is found in this list it is returned as it is else it is fetched from the disk. Periodically the "dirty pages" in the free list are looked for in the free list and are then batch persisted back on to the disk. The use of free list to some degree implements LRU like policy because if suppose a page got evicted from the main FIFO list and if it has not been a long time since it got evicted. Now if a reference for this page comes then chances are high that we would be able to retrieve it from the free list directly instead of reading the entire page from the disk.

Actually: the system maintains a modified list, a standby list, a zroed list and a free list. But the main idea remains the one that is illustrated above. Here is what actually happens. "In an effort to keep a minimum number of pages available to the system at all times, the VMM (Virtual Memory Manager) moves pages from either the Modified or Standby list to the Free list. Modified pages must be written to disk first and then marked as Free. Standby pages do not need to be written because they are not dirty. Free pages are eventually zeroed and moved to the Zeroed list. Pages in the Free and Zeroed lists are immediately available to processes that request pages of memory. (Each time a page is moved from one list to the next, the Virtual_memory Manager updates the page-frame database and the PTE for the page. It is important to note that pages in either of the transition states are literally in transition from Valid pages to Free pages.)"

5a) Internal fragmentation should increase In paging scheme frames are allocated as units. Internal fragmentation occurs when the memory requirements of a process do not happen to coincide with page boundaries which results in a partially full last frame that is allocated. e.g. In the worst case a process would need n pages plus 1 byte. It would be allocated n+1 frames, resulting in an internal fragmentation of almost an entire frame. In general, we expect the internal fragmentation to average one-half page per process. By this consideration we can see that increasing the virtual memory page size in a pure paging scheme would increase the amount of internal fragmentation.

External fragmentation should remain the same. Since, in a pure paging scheme there is no external fragmentation as any free frame can be allocated to a process that needs it.So even by increasing the page size there should still be no external fragmentation.

5b) Internal Fragmentation should increase. External fragmentation should come down to nearly zero.

Pure segmentation suffers from external fragmentation only and pure paging suffers from internal fragmentation only. So converting from a pure segmentation to a pure paging scheme should lower the external fragmentation to zero and should increase the internal fragmentation.

6a) FALSE. Lottery scheduling is not suited for real-time system because of its probabilistic nature. There is no way for a lottery scheduler to ensure that a job will be accomplished by any given deadline. (Eventually, if a thread does not possess all the tickets, the probability that it will not be terminated by a given time T is strictly positive for any T)

6b) FALSE. The use of lock ranks at the OS level allows to detect deadlock situations involving locks since it detects circular wait conditions, one of the four necessary conditions for a deadlock to occur. But it cannot prevent such deadlocks. Nevertheless, by killing threads responsible for such situations, it is possible to escape deadlocks at the expense of loosing state.

6c) TRUE. Perfect scheduler decisions can enforce critical sections without locks by scheduling threads accordingly to their access pattern to shared resources.

7) False sharing occur when processors write to a shared page but not at the same location. Simultaneous updates of individual elements in the same page coming from different processors invalidates entire pages, even though these updates are logically independent of each other. Each update of an individual element of a page marks the page as invalid. Other processors accessing a different element in the same page see the page marked as invalid. They are forced to fetch a fresh copy of the page from memory, even though the element accessed has not been modified. This is because coherency is maintained on a page basis, and not for individual elements.

Increasing the page size will increase the probability that two processors use the same page to store logically independent elements and will therefore result in more false sharing. As a result there will be an increase in interconnect traffic and overhead.