**CS 33211** # Homework #3 **OS**

**Due in class at 12:30pm on Wednesday 15 November 2006**
*typed answers preferred*

---

**1. Static relocation is usually accomplished by having software (the loader in the OS) modify addresses when a program is loaded into memory, while dynamic relocation is usually accomplished by having hardware (the MMU) modify addresses as the program executes. What are the tradeoffs of these two techniques?**

With static relocation, all addresses in the program are modified when the program is loaded, which means that if it must later be moved to another location all those addresses must be modified once again. In contrast, with dynamic relocation all that must be modified is the base location of the process, making it much easier to move a program.

With static relocation, if an addressed is referenced multiple times it is only modified once (this is good), but all addresses are modified regardless of whether of not they are referenced (this is bad). In contrast, with dynamic relocation addresses are modified only if they are referenced (this is good), if an address is referenced multiple times it is modified multiple times (this is bad).

Many other answers possible: dynamic relocation provides more protection than static relocation, dynamic relocation has more hardware overhead while static relocation has more software overhead, …

**2/3. When a process is loaded into memory, a best guess is made by the OS as to how much memory space to allocate for the stack and heap. If insufficient memory space is allocated for these two and more space is needed, how is this problem handled with partitioning, with segmentation, and with paging? (*this question counts double*)**

With partitioning, the process must be allocated to a bigger partition, either by growing the current partition or moving it to a larger one. If no larger space is available, the system may have to swap some processes out or compact segments to create a larger free space.

With segmentation, if the stack and heap are in separate segments, all that is necessary is to grow the size of that segment, possibly moving to larger space elsewhere in memory. This is similar to what is done with partitioning, but only the one segment is affected, not the (much larger) entire process.

With paging, the process would have to be allocated more pages and the stack or heap moved into those new pages as appropriate. Since there isn't a direct correlation between the stack/heap and specific pages this is more difficult than with segmentation but less so than with partitioning.

4. **A paging system typically uses a bit vector to represent the frames and find the first free frame. How is this better than simply using a linked list to keep track of free frames?**

   Using one bit per frame takes far less memory space than would be required for the pointers in a linked list. Further, with hardware support to scan a bit vector to find the first free bit, the one instruction required for that scan would be much faster than the many instructions required to search the linked list for a free frame.

5. **In what way is demand paging similar to caching in a CPU's instruction/data cache, and in what way is it different?**

   In demand paging, the physical memory is essentially a "cache" of the combined virtual memories of all processes in the system. Like any caching system, it is much faster to access the cache than the data elsewhere, but the cache is far smaller than storage elsewhere, so the system much choose carefully what data to keep in the cache. Like a CPU's instruction/data cache, demand paging takes advantage of the Principle of Locality of Reference, and uses some policy (i.e., page replacement) to specify which objects in the cache are replaced by new objects being cached.

6. **Does the working set model of page allocation support global or local replacement of frames? Explain.**

   Local page replacement — in general, a process is allocated enough frames for its working set, and page replacement is done from those frames. However, the working set size changes over time, and as it does so, the number of frames allocated to a process may change, so though it does not directly support global replacement (i.e., letting a process immediately take one of the frames of another process), working sets do provide some of the advantages of global replacement by letting processes that need more frames get more fames (though not necessarily immediately).