

ITEE



What's New:

Questions this time build on the last set of classroom exercises. There is some repetition for those who missed the lecture, but the intent is that you do that part quickly and focus on the new content.

Learning objectives for this week:

- Understand the major concepts we've covered, including:
 - why memory management is needed
 - differences between segmented and paged models
 - how paging is implemented
- Apply these concepts to a specific situation, to understand how memory is being managed
- Apply these concepts to choosing an appropriate strategy for a given situation
- Understand where and how different strategies apply, and their significance in real systems

As before, you should aim to answer all of these questions by the end of the course. Being able to answer them by the end of week 8 is a useful goal. Be prepared: if you know what you can't do yourself, you will make most efficient use of the tutorial.

1. Concepts

- a. Why is non-contiguous memory allocation desirable? Consider what would happen otherwise if you had to load several programs into memory, and load new programs into memory after others finished executing.
- b. A segmented system corresponds to a programmer's view of memory whereas a paged system does not. True or false? Explain your reasoning.
- c. Page table design is a matter of trading memory for speed. Discuss this statement, using examples from the lectures.

2. Relocation

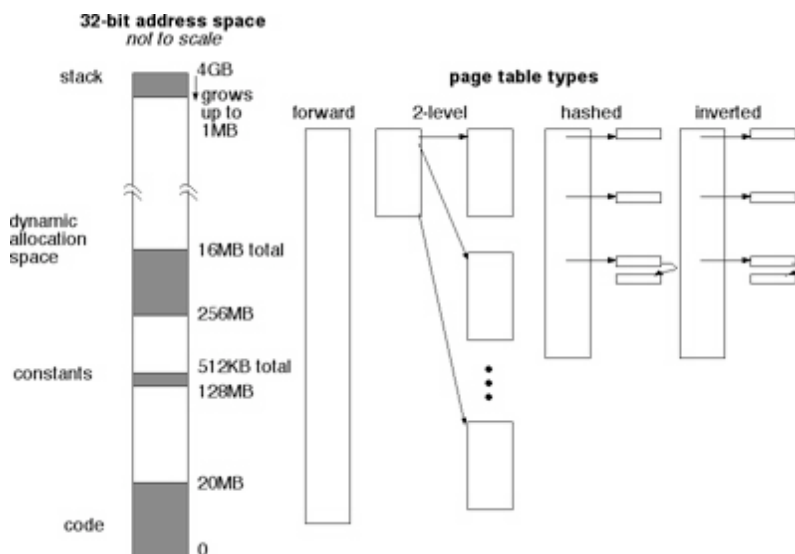
- a. When a disk produces data as a consequence of a disk read, it needs to write the results to a location in memory where it knows the absolute address. Explain why this may cause problems for any scheme which allows relocation, and how these problems can be solved.
- b. The original Macintosh design did not include allowance for multitasking. Its memory management scheme implemented memory compaction in a way which required a programmer to be aware of when compaction may or may not occur. Why is this design poor in a multitasking environment?
- c. Is relocation important in a page-based scheme (considering only the information in Chapter 9)? Explain your answer.

3. Address Binding

- a. In each situation, explain whether the time binding occurs is at compile, load or run time (as defined on p 275 of the textbook):
 - i. A page is not allocated in physical memory until it is first used.
 - ii. A page is allocated to a physical frame when the program is about to run, and once allocated stays in the same location until execution terminates.
 - iii. The operating system allows a user program to see the value of a global variable at a specific, invariable location, and the compiler is permitted to use the name of this variable, but the actual physical location is only inserted into the code when it is linked.
- b. Early binding is easy, while late binding is flexible. Discuss this statement, in the light of any examples you know (including programming languages).

4. Page Table Implementation

Under the following assumptions (with a potentially helpful picture):



- 4KB pages
- 512MB of physical memory
- for 2-level paging, half the page number is used for each level

- page table entries are rounded up to a whole number of bytes
 - a hashed page table is 80% full
 - an inverted page table has one entry per physical address, and needs 25% more entries for handling hash collisions
- a. What fraction of real memory does the process need (excluding OS overheads like page tables)?
 - b. Work out the size of a forward page table (done in Week 7 classroom exercises).
 - c. Work out the size of a 2-level page table (done in Week 7 classroom exercises).
 - d. Work out the size of a hashed page table (done in Week 7 classroom exercises). Make sure you take into account extra information which would need to be stored in the page table, to identify whether a particular virtual page had actually been found.
 - e. How would an inverted page table differ from a hashed page table in terms of
 - i. the situation where there is more than one process in memory?
 - ii. total memory usage? (Think of what else is needed to handle more than one process in using the same page table.)
 - f. IBM's RS/6000 RISC system used an inverted page table design, in which the low 13 bits of the page number uniquely determine the location in the page table. In other words, given that the hash function has taken you to a particular location, you know 13 bits of the virtual address. Why do you think they have implemented their inverted page tables this way?
 - g. What would work well as a hash function for the hashed and inverted page tables?
 - h. Which strategy is best in terms of page table size, in this situation? Which would be best if you had many processes in memory at the same time?
 - i. Now, redo the previous parts of the question assuming that memory is allocated in a large number of small, randomly scattered objects, each less than a page in size. Think through the differences, rather than redoing the numbers in detail. In this scenario, would your idea of the best strategy change?