Memory Paging

Paging and segmentation are implemented in all modern operating systems.

Definition

Memory allocation scheme that allocates multiple non-contiguous ranges of memory per process

Base1 + limit1; base2 + limit2; etc.

Basic Theory

- Hardware for speed
- Address translation by MMU
- Translation page table
- CPU addresses are logical
- MMU addresses are physical

Memory is organized into a series of frames known as pages. The translation page table is the key. The MMU maintains this internally. The CPU, then, generates a logical address for its access to memory. Rather than stating memory location 8000, it says "I want to access memory location 4 away from my base." The MMU then translates this to an actual physical address in memory.

Organize memory like pages in a book. Allocate memory randomly. If the process needs 24 pages, allocate 24 random pages. Maintain a table so that the translation can be done.

On the average, when we have internal fragmentation, it is ½ page, 50%.

Logical Addresses

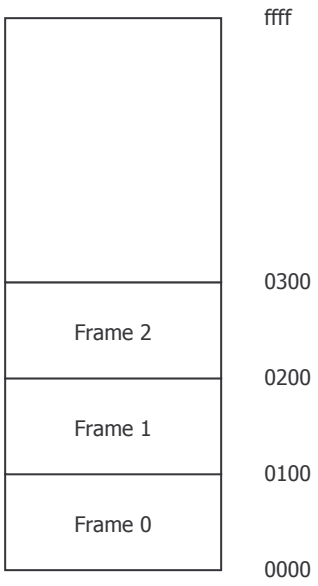
Page Number	Byte Offset
p	d

The majority of compilers already create a logical address structure. To the program, its memory begins at 0.

Physical Address

Frame	Byte Offset
f	d

In the example of the HC11, consider the first byte to be the frame.



Frame size and page size are equal. Regularity implies a simpler circuit and therefore smaller. Smaller implies cheaper and faster.

In general, algebraically, we can begin to discuss sizes of things in memory. If you have an m-bit memory address, you will also have an m-bit logical address, because the memory space is the same.

$$2^m \text{ bytes of memory}$$

$$2^n \text{ frame size}$$

Rule of thumb: an 8kb frame is a pretty good frame size and will help to limit internal fragmentation. The Intel Pentium supports 8 kb or 4 kb frames. It is a complete dichotomy.

$$\frac{2^{32} \text{ total bytes}}{2^{13} \text{ bytes per frame}} = 2^{m-n} = 2^{19} \text{ frames}$$

$$p = f = m - n \text{ bits long}$$

These virtual pages are mapped to physical memory addresses.

Fragmentation Comments

There is no external fragmentation. External fragmentation occurs when there is not a contiguous region of memory large enough to hold the process.

However, there is internal fragmentation that is, on the average, of the size $\frac{1}{2} f$. The worst case internal fragmentation occurs when only one byte of a frame is used.

To decrease internal fragmentation, you decrease the frame size. However, in this instance, the number of frames goes up. Therefore, the table sizes grow.

Large # frames → slower lookup
Small # frames → very fast lookup, but higher internal fragmentation

Modern systems traditionally support 8 kb as a frame size. Frame sizes from 512 kb to 1 mb are found on some architectures.

Loading

Determine process size. Calculate number of pages required. Then, go through the frame table which frames are available and determine which frames are available an

In general, the page table information is stored in a backing storage device and is accessed by the OS at a context switch.

The MMU is a device controlled by the operating system.

Current MMU page table must be saved to backing storage device.
New page table is loaded to the MMU.

Page Table Options (in MMU)

1. Privileged-access registers (for small number of pages)
    a. Extremely fast. Privileged-access so that a user process could not modify them
2. Memory or backing storage device
    a. MMU would need some sort of pointer register to the base of the process page table.
    b. If you have to dereference a pointer to get information, that's a first memory access. You then go out to memory to get the info off the page you wanted. You now have two memory accesses, and that is a penalty.
    c. Access delay is no longer acceptable
3. Translation look-aside buffer (TLB)

Translation look-aside buffer (TLB)

Very fast cache memory implemented as part of the MMU. This is an associative cache memory. It is presented with a key, which is hashed very rapidly to the result.

It is worth the effort and expense it costs to create the associative cache because of the speed advantage it gives.

Performance

Hit ratio, *H*, is the key
% of time that we actually access this TLB and find our page/frame pair. The goal is to make the ratio high.

$$effective\ time = H \cdot t_{access} + 2(1-H) \cdot t_{access}$$

Miss
When there are misses (page/frame pair not found in cache), we have to load the cache from the memory.

Replacement strategy.

Results in two memory accesses. Do a look up, realize it's not there, re-load cache from memory, do another lookup.

Hit ratios of 75-80% are typical.

Principle of data locality – people declare variables and access them in loops, near where they were declared. Then, they're thrown away.

If you've accessed _this_ location in memory, you will probably access one nearby in successive lookups.

MMU Protection Bits

There are two types of protection bits:

- Memory protection bits
  - Allow the process page table to be read, written, read/write, or executed
  - The compiler and the loader are responsible for setting page bits appropriately
- Valid-invalid bits