Physical memory management

Memory management is the most important part of a young kernel. Separating the two concepts of physical and virtual memory management is crucial to creating a decent kernel.

Virtual memory management is extremely useful. It means that compilers can produce a program that relies on the code being at an exact location in memory, every time it is run. But for the virtual manager to work, it needs to be able to allocated physical pages/frames in the RAM. Here is where the physical memory manager comes to play. He will allow the kernel to allocate physical pages that can be mapped to a virtual address, i.e. a frame for the page.

Memory management is a close relationship between these two managers:

* The physical memory manager (PMM), which knows which areas of memory are free to use and which are allocated.
* The virtual memory manager (VMM), which manages the mappings between virtual and physical addresses.

For stability reasons the PMM shouldn’t rely too much on the virtual memory manager, only the virtual manager should rely on the physical manager. This can be achieved by some nifty coding, for example you could find the end of the kernel using a linker script. Behind it you’ll find some place for the physical manager. Then you should let the manager protect its data, and afterwards it should ask the virtual manager map it so it can be used.

There are a number of ways to manage physical memory, but there are 2 major models:

* Stack: We keep a stack of all the addresses and we push and pop pages from that stack.
* Bitmap: We keep a large array of 1 and 0 which tell us if a page has been allocated.

The stack implantation is fast but it takes up some memory, you can’t pop a specific address and it can be crashed if a page is pushed twice on the stack, i.e. it’s unstable. The bitmap implementation is slow but you can allocate a specific address, it doesn’t take up too much memory and it’s very stable. Pro-Typ3 uses a hybrid model, which implements both models. This is very stable and fast, but it isn’t very memory efficient, for 4 GB it will take 4, 125 MB, which is actually just 0.1% of the total memory and only 3.1% bigger than the stack model but it’s still quite a bit. It is also a bit slower than the stack model, but only by a few instructions.